Unsupervised Learning on Social Data

Felix Borutta

München 2019

Unsupervised Learning on Social Data

Dissertation
an der Fakultät für Mathematik, Informatik und Statistik
der Ludwig–Maximilians–Universität München

vorgelegt von
Felix Borutta
aus München

München, den 25.10.2019

Erstgutachter: Prof. Dr. Peer Kröger
Zweitgutachter: Prof. Dr. Emmanuel Müller
Datum der mündlichen Prüfung: 11.03.2020

Eidesstattliche Versicherung

Hiermit erkläre ich, Felix Borutta, an Eides statt, dass die vorliegende Dissertation ohne unerlaubte Hilfe gemäß Promotionsordnung vom 12.07.2011, § 8, Abs. 2 Pkt. 5, angefertigt worden ist.

München, 25.10.2019

....

Felix Borutta

Contents

Abstract

Analyzing social data comprises huge potentials for companies. On the one hand, enterprises can massively benefit from identifying market segments to subsequently apply targeted marketing strategies. Another common task in this area of application is to identify so-called influencers, i.e., hub users within communities, that promote new products so that companies can launch these products into the market quickly. On the other hand, the analysis of social data gathered from the employees of a company itself can significantly improve the internal organization by revealing insights that might enable collaboration opportunities or an optimized exchange of information. Therefore, social data analysis has become an important component of the analytics strategies in many companies. However, it is obvious that this requires not only the raw data but also the appropriate tools for analyzing the data. This thesis addresses several recent research questions arising in the field of social data analytics. Though the proposed algorithms are suitable for solving problems that are highly related to social data analysis, most of them generalize to other domains, as well.

Given the attribute information, i.e., age, profession, interests, etc., of the users from a social media platform, it is often difficult to identify specific user groups in these data. Two common reasons are the large number of dimensions that the data might have and the fact that social media data is generally dynamic. Data mining approaches like subspace clustering algorithms are established methods to identify groups of objects that are, wrt some distance metric, similar in certain subspaces. A special kind of subspace clustering is correlation clustering which aims at identifying arbitrarily oriented linear subspaces, rather than being limited to only detect axis-parallel subspace clusters. However, most previous approaches work on static data and cannot deal with data which might change over time. To this end, this thesis presents two novel algorithms for correlation clustering on streaming data. While one relies on covariance matrix factorization, the other algorithm uses Hough transformation for identifying correlation subspaces. Moreover, this thesis presents two further data mining methods which both are based on microblog data. The first is a hierarchical clustering algorithm that relies on text mining to create social maps at different scales by learning a clustering model based on the most prominent discussed topics in certain spatial areas. The second is an algorithm that is suitable to identify yet unknown users by classifying previously mined spatio-temporal mobility patterns.

Another research direction addressed in this thesis is learning based on network data. Since users are usually related to each other through friendship relations or interactions, social data is often modeled as a graph where users are considered as vertices and friendship relations, resp. interactions, form links between two users. The proposed methodologies for learning vertex representations all make use of the information captured in the topology of the resulting graph. Two of the presented node embedding algorithms only rely on topological information but learn node representations by following different goals. One is based on the so-called homophily assumptions and tries to learn similar vector representations for nodes that are close-by in the graph. The other aims at creating similar node descriptors for nodes that have a similar functionality within the network, e.g., two influencers in a social network should get a similar vector representation even if they are not close-by in the graph. A further algorithm that uses the latter type of node embeddings alongside a suitable, unsupervised aggregation method finally enables to classify entire graph structures based on the functionalities that the nodes within a graph structure have. Furthermore, this thesis also presents a semi-supervised approach for learning node representations by incorporating node label information from the local neighborhoods beside of topological information. Finally, aiming at enabling graph fusion, we also propose a graph neural network based model that is able to learn matchings of nodes from two partly overlapping graphs.

Zusammenfassung

Die Analyse sozialer Daten stellt ein großes Potential, sowohl für die interne Organisation von Firmen, als auch für die wirtschaftliche Verwertung von Produkten dar. So können Unternehmen zum Beispiel anhand der Analyse sozialer Daten bestimmte Marktsegmente identifizieren, um anschließend auf die Märkte zugeschnittene Marketing Strategien einzusetzen. Ein weiteres Beispiel bei dem die Analyse sozialer Daten wirtschaftlich erfolgreich eingesetzt werden kann ist die Analyse interner Strukturen und Abläufe. Kennt ein Unternehmen diese, kann es diese optimieren, um so die Wirtschaftlichkeit und letztlich die Produktivität zu steigern. Hinsichtlich dieser und weiterer Beispiele ist die Analyse sozialer Daten bereits eine wichtige Komponente in vielen Analysestrategien von Unternehmen weltweit geworden. Um diese Strategien umzusetzen bedarf es allerdings nicht nur der Daten, sondern insbesondere auch geeigneter Methoden zur Analyse. In dieser Arbeit werden daher verschiedene Problemstellung im Bereich der Datenanalyse mit Bezug auf soziale Daten mittels wissenschaftlichen Methoden untersucht und Lösungsvorschläge erarbeitet. Obwohl sich die vorgestellten Algorithmen für die Analyse sozialer Daten eignen, sind sie dennoch so generisch, dass sie prinzipiell auch in anderen Anwendungsgebieten einsetzbar sind.

Im Allgemeinen ist das Identifizieren verschiedener Nutzergruppen anhand der Daten die ein Nutzer in sozialen Netzwerken angibt nicht offensichtlich. Das liegt zum Einen an der Menge der Informationen, und zum Anderen daran, dass sich diese Informationen häufig ändern. Betrachtet man die, aufgrund der Menge von Informationen, hohe Dimensionalität von sozialen Daten sind Data Mining Ansätze wie Subspace Clustering Algorithmen bewährte Methoden, um Nutzergruppen hinsichtlich bestimmter Untermengen der Informationen zu identifizieren. Eine spezielle Art des Subspace Clustering ist das sogenannte Correlation Clustering. Anstatt das Identifizieren von Gruppen auf gegebene Merkmale zu beschränken, sogenannte achsen-parallele Unterraumcluster, erlauben es diese Ansätze Gruppen auch anhand kombinierter Merkmale, also in beliebig orientierten Unterräumen, zu erkennen. Bisherige Methoden zum Erkennen von beliebig orientierten

Unterraumclustern funktionieren allerdings vorrangig auf statischen Daten und können oft nicht auf Daten angewendet werden, die sich über die Zeit ändern. Aufgrund dieser Beschaffenheit werden in dieser Arbeit zwei Algorithmen präsentiert welche dies ermöglichen. Während einer der Algorithmen auf einer inkrementellen Version der Hauptkomponentenanalyse basiert, nutzt der Andere Hough Transformationen, um beliebig orientierte Untercluster zu finden. Des Weiteren werden in dieser Arbeit zwei Data Mining Methoden zur Analyse von Microblog Daten präsentiert. Die erste Methode erstellt aus Microblog Daten Bewegungsmuster, die später zur Identifikation von Nutzern genutzt werden, und die zweite Methode ermöglicht es mittels Text Mining und hierarchischem Clustering soziale Landkarten zu erstellen.

Eine andere Forschungsrichtung, der sich diese Arbeit widmet, ist das maschinelle Lernen mit Netzwerkdaten. Da Nutzer im sozialen Kontext typischerweise miteinander in Beziehung stehen, sei es durch Freundschaftsbeziehung oder durch menschliche Interaktionen, bietet es sich häufig an soziale Daten in Form von Netzwerken darzustellen. In den resultierenden Graphstrukturen bilden Nutzer folglich die Knoten und Freundschaftsbeziehungen oder Interaktionen werden als Kanten abgebildet. Die im zweiten Teil der Arbeit vorgestellten Methoden machen explizit Gebrauch von diesen Beziehungsinformationen, um vektorielle Repräsentationen von Nutzern zu lernen. Zwei der vorgestellten Methoden beschränken sich auf das Nutzen der im Netzwerk vorhandenen topologischen Eigenschaften um Knotenrepräsentationen zu lernen, allerdings jeweils mit einem anderen Ziel. Während eine Methode auf der sogenannten Homophilie-Annahme basiert und versucht Nutzer so in einen kontinuierlichen Vektorraum einzubetten, dass Nutzer die im Netzwerk in enger Beziehung stehen auch im Vektorraum nahe beieinander liegen, lernt die andere Methode anhand struktureller Eigenschaften. Das bedeutet, dass Letztere die Knoten so in den Vektorraum einbettet, dass Nutzer, die ein ähnliches Beziehungsgefüge pflegen am Ende im Vektorraum nahe beieinander liegen. Eine weitere Methode die in dieser Arbeit vorgestellt wird nutzt Struktur-basierte Knotenrepräsentationen um ganze Graphstrukturen in einen Vektorraum einzubetten, so dass die resultierenden Vektorrepräsentationen letzten Endes benutzt werden können um Graphklassifikatoren zu lernen. Außerdem wird ein semi-supervised Ansatz präsentiert, welcher teilweise vorhandene Informationen zu Knotentypen für das Erlernen eines Klassifikators für Knoten ausnutzt. Zum Abschluss der Arbeit wird schließlich noch eine Methode zum Fusionieren zweier Graphen, inklusive konkretem Anwendungsfall, nämlich dem Fusionieren von Straßenkarten, vorgestellt.

Chapter 1

Preface

1.1 Introduction

Social data, for instance, data gathered from social media, interactions on social media platforms, or interactions between mobile devices in the real world generally comprise a plethora of different kinds of information. Among others, these types of information may include personal information of the users (e.g., age, profession, place of residence, etc.), specific interests of the users (e.g., what music a user likes or what kind of books she reads), relational information (e.g., friendship relations among users or interactions among users) or even geo-spatial information (e.g., information about specific places that a user visited). However, given these different kinds and quite often huge amounts of data, it is almost impossible to generate useful knowledge from it without having suitable methods for data analysis. This is reasoned by the facts that firstly, the data is usually high dimensional and secondly, that there typically exist complex patterns and relations, i.e., so-called *hidden information*, within the data that humans cannot reveal by only “looking” at the data.

Considering the example of friendship recommendation within a social network for instance should help to understand the problem. Given the social network where vertices are users and links represent real-world friendship relations, and additionally some user attributes, e.g., age, profession, favorite movie genre and type of vehicle, associated with each user. The task is to recommend links between users that have not been connected, yet. A straightforward approach would be to compare the user attributes with respect to some similarity measure and recommend links between similar users given that those links do not exist, yet. At first glance this may totally make sense since people in about the same age with similar interests are obviously more likely to become friends than differently aged persons who do not share any interests. However, this does not reflect the reality. Following Tobler's first law of geography which states that "everything is related to everything else, but near things are more related than distant things" [248], the described approach cannot make the "desirable" recommendations since some user coming from a completely different part of the world might be very similar (in terms of age and interests) to a query user, but due to the large distance between the two users, it is very unlikely that they will actually become friends. However, what also can be derived from Tobler's law is that the network topology of the social graph should encode some information about spatial proximity, although the links are supposed to only encode friendship relations. Precisely, it can be assumed that members of a community within the social graph are all located near each other. Though it was possible to derive this knowledge based on expertise, the proximity information that is encoded is some kind of *hidden information* which is not easy to detect by non-experts or humans that only "have a look at the data". Note that there might be other kinds of hidden information, even in the this small example, that are even more difficult to reveal.

In short, the goal of social data analysis can be summarized as follows: given the social data, social data analysis aims at generating knowledge about the real world from information that is explicitly and implicitly modeled within the data.[1] This thesis presents several different approaches to gain knowledge from data, compress this knowledge in low-dimensional vector representations and partly use these representations to solve subsequent tasks like link prediction. Throughout the thesis, applications from the area of social data analysis are taken as motivating examples but in general and as is shown by the experimental evaluations of the methods, the presented algorithms generalize to data from a variety of other domains as well.

Technically, all of the proposed methods are positioned in the field of machine learning. In general, machine learning algorithms can usually be classified into two basic categories: supervised and unsupervised methods.[2] Supervised methods are primarily characterized by the property that these algorithms learn a function that maps an input vector to some desired target or output vector [39]. By adjusting the model parameters with respect to the provided input-output pairs that are passed to the model during training, the trained model is finally able to correctly transform or classify yet unseen input signals. In particular, this means that the machine requires to know

[1]This is not a goal which is exclusive for social data analysis. In fact, this is a general goal of data-driven analysis in any domains.

[2]Note that there exists a mixture of both which is called semi-supervised learning. These techniques typically combine both paradigms.

the target vector in advance for the learning phase. These target vectors are referred to as ground truth throughout the thesis. In contrast to supervised learning methods, unsupervised methods do not require such a ground truth. The key idea behind unsupervised methods is that the machine learns the ability to detect and distinguish different patterns in the data without having the kind of "teaching mechanism" as is used for supervised methods. The key difference of the intuitions for both supervised and unsupervised learning can easily be described by using an example of how humans learn. Consider a child that is learning to recognize the letter m for instance. Obviously, as the letter m may appear differently in different fonts and handwritings, the child needs some examples accompanied with the information that the examples are ms to learn how an m looks like. Once internalized, the child will be able to recognize the letter m, even from a yet unseen font. In machine learning terminology, this is called supervised learning. On the other hand, consider the case where the child needs to distinguish between the two letters e and i. For this task, the child does not need to know the exact appearances of e or i. In fact, it just needs a couple of observations of the letters e and i to gain the experience that they are two different *concepts*, e.g., that they just look differently. In terms of machine learning, such kind of learning is referred to as unsupervised learning. Somewhat more formally, given several instances of a random input vector $\mathbf{x}$, each associated with its corresponding ground truth vector or value $\mathbf{y}$, supervised learning typically aims at learning the probability distribution $p(\mathbf{y}|\mathbf{x})$. In contrast, unsupervised learning has the goal to learn the probability distribution $p(\mathbf{x})$, or at least important properties of it, only based on some instances of a random input vector $\mathbf{x}$ [101].

As the main part of this thesis focuses on unsupervised learning techniques, Section 1.2 provides a brief overview of relevant fundamentals of unsupervised learning in general. Section 1.3 then highlights the contributions and surveys the structure of this thesis.

1.2 Fundamentals

Although it might seem more intuitive to train a model based on ground truth, unsupervised learning encompasses a couple of advantages compared to supervised learning methods. The biggest advantage is that unsupervised methods are typically more broadly applicable since they obviously do not require labeled data. In fact, many applications suffer from the lack of ground truth data and generating labels is generally expensive. It often requires knowledge from domain experts and, depending on the complexity of the task, supervised methods may also need a remarkable amount of labeled data. Two common approaches to overcome this issue by using unsupervised methods are (1) to train a model on large amounts of unlabeled data and later inject supervision into the outcome of the model, or (2) to use only small amounts of labeled data to train a basic model which is later tuned by using an unsupervised training procedure with larger amounts of data on top. Another benefit of unsupervised methods is their ability to greatly support data exploration. For instance, finding groups of patterns can provide valuable information for further data processing steps. Furthermore, some unsupervised methods have the advantage that they can cope with slowly evolving data, i.e., concept drifts. This might be particularly useful for tasks where temporal trends play a key role, e.g., identifying users in a social network that temporarily have a high influence on other users in the network. Using supervised methods, one would have to re-train, respectively train a new model, with recent ground truth data to keep an up-to-date model [84].

This thesis concentrates on three very fundamental tasks of unsupervised learning, namely

- clustering,
- feature extraction from unstructured data, and
- representation learning,

which are introduced in the following sections.

Clustering

Clustering denotes the task of grouping objects of a data set such that objects within a group are similar with respect to some similarity measure while objects that do not belong to the same group are dissimilar. Typical usages of clustering are data exploration to get insight into the nature of the data or preprocessing for other algorithms. There exists a variety of different clustering algorithms that in turn follow different concepts which all significantly vary in terms of properties they model. The clustering concepts that are used in this thesis are:

- **Partitioning clustering.** Given a dataset $\mathcal{D} = \{x_0, \dots, x_n\}$, a partitioning clustering algorithm aims at finding a partitioning $\mathcal{C} = \{C_1, \dots, C_k\}$ such that each object x_i is assigned to exactly one partition C_j with $0 \leq i \leq n$ and $1 \leq j \leq k$. During the assignment phase, those algorithms typically optimize for some objective function. A well-known representative of this class of clustering algorithms is *k-means*, i.e., an iterative partitioning procedure where the objects in $\mathcal{D}$ are assigned to clusters $C_j \in \mathcal{C}$ with the objective that the sum of distances between data objects and corresponding cluster centers is minimized. Note that cluster centers are initially chosen according to some initialization strategy. After each iteration, the cluster centers are updated, e.g., by setting the centroid to the mean of all data objects that have been assigned to the corresponding partition, and the assignment phase restarts. The partitioning procedure terminates if the cluster centers converge to stable positions.

- **Density-based clustering.** Methods of this group of clustering algorithms define groupings based on the local densities of the data space. Precisely, regions that are dense with respect to some distance measure are considered as clusters, which in turn are separated from each other by sparse regions. The state-of-the-art approach in this class of algorithms is *DBSCAN*, i.e., a method that relies on local point densities and spatial reachability criteria to define the final clustering. Given a dataset $\mathcal{D}$, a distance threshold ϵ and a density threshold *minPts*, *DBSCAN* searches for so-called core objects $s \in \mathcal{D}$ for which it holds that $|\mathcal{N}_\epsilon(s)| \geq minPts$, with $\mathcal{N}_\epsilon(s) = \{o \in \mathcal{D} | dist(o, s) \leq \epsilon\}$ being the ϵ-neighborhood of s. Once such a core object is found, a new cluster, initially containing the core object itself, is created and extended by finding density-reachable objects, i.e., objects that are within the ϵ-neighborhood of density-reachable core objects. If no more density-reachable object can be found for the current cluster, the algorithm restarts by searching a new initial core objects from the yet unprocessed objects in $\mathcal{D}$. Noteworthy, an important property of this procedure is that some objects may not be assigned to any cluster due to its low local density. Such objects are marked as *noise*.

- **Subspace clustering.** Roughly spoken, subspace clustering aims at identifying clusters in lower dimensional subspaces rather than the full high-dimensional space. The intuition behind subspace clustering is that data is often high-dimensional, i.e., the number of features is large, and for several reasons, e.g., the curse of dimensionality or the fact that different features are differently relevant for various clusters, a large body of established methods cannot cope properly with the large number of dimensions. By identifying various subspaces in which data objects form groups, subspace clustering algorithms generally provide valuable insights and overcome certain issues that appear when performing comparable global feature reduction as preprocessing for subsequent data mining tasks. In general, the term subspace clustering represents a whole family of different algorithms that all have slightly different underlying assumptions and goals. While axis-parallel subspace clustering algorithms reveal clusters only in spaces spanned by a subset of the originally given features, some subspace clustering techniques are able to detect clusters in arbitrarily oriented subspaces. The latter approaches are called correlation clustering algorithms. Note that the literature sometimes also differs between projected clustering where data objects are assigned uniquely to subspace clusters and subspace clustering where data objects may belong to multiple subspaces. However, in this thesis, the term subspace clustering is used globally to refer to the entire family of algorithms.

- **Hierarchical clustering.** In many applications data may contain clusters that are nested within other clusters, i.e., there might exist an internal hierarchy of groupings. Common clustering approaches usually cannot reveal such hierarchical relations due to aiming at some global objective (like *k-means*) or relying on a global parameter setting (like *DBSCAN*). Hierarchical clustering approaches have been developed to overcome this issue. In general, there are two classes of hierarchical clustering approaches: agglomerative clustering and divisive clustering. While agglomerative clustering methods assign each data object to its own cluster and greedily merges pairs of clusters, divisive clustering approaches initially assign each object to one cluster and recursively performs splits until each object is its own cluster. Either way the result is a strict hierarchy of clusters. The canonical procedure for agglomerative clustering is as follows: each data object in database $\mathcal{D}$ initially forms its own cluster X_i. After precomputing all pairwise distances between the clusters with respect to some distance function $dist_c(X_i, X_j) = min_{x \in X_i, y \in X_j} dist(x, y)$, the algorithm iteratively merges the closest pair of clusters (X_i, X_j) by forming a new cluster $X_k = X_i \cup X_j$, substitutes the concerned clusters X_i and X_j with the newly formed cluster X_k in the set of considered clusters, and computes the pairwise distances between the new cluster and all other currently considered clusters. This is repeated until there remains only a single cluster consisting of all data objects. Note that the closest pair of clusters requires some definition of similarity between clusters that may vary between various agglomerative clustering strategies.

- **Graph-based clustering.** Given a graph $\mathcal{G} = (V, E)$ with V denoting the set of vertices and E being the set of edges that connect pairs of vertices, graph clustering algorithms generally aim at clustering groups of vertices such that the number of edges within a cluster of vertices, i.e., internal edges, is notably greater than the number of edges that connect different clusters, i.e., external edges. Formally speaking, given a clustering $\mathcal{C} = \{C_1, \dots, C_k\}$, with $C_i = V \setminus \bigcup_{j \in \{1,\dots,k\}, j \neq i} C_j$ for $i \in \{1, \dots, k\}$, the general clustering objective is to define a partitioning that ensures a high average internal density, defined as

$$\delta_{int}(\mathcal{C}) = \frac{1}{k} \sum_{i=1}^{k} \frac{|\{e_{uv} | e_{uv} \in E, u \in C_i, v \in C_i\}|}{|C_i|(|C_i| - 1)},$$

while the external density, defined as

$$\delta_{ext}(\mathcal{C}) = \frac{|\{e_{uv} | e_{uv} \in E, u \in C_i, v \in C_j, i \neq j\}|}{|V|(|V| - 1) - \sum_{l=1}^{k} (|C_l|(|C_l| - 1))},$$

i.e., the ratio of internal edges to the number of all possible external edges, is relatively low [225].

Further prominent clustering concepts, which are not used throughout the thesis, but mentioned for the sake of completeness as they might be of interest for further refining some of the presented methods are distribution-based clustering algorithms, e.g., the *expectation maximization* algorithm, or neural approaches, e.g., *self-organizing maps*.

To summarize, Figure 1.1 visualizes the clustering results achieved on four small datasets with *k-means*, *DBSCAN*, *CASH*, i.e., a method for correlation clustering, and agglomerative approaches with proper merging strategies, i.e., merging the clusters by minimizing the variance of the clusters, respectively for the two moons dataset, merging the clusters with minimum distance between all objects of the two considered clusters. Each of the clustering algorithms follows a different goal in terms of grouping objects and therefore they perform differently "good" on the used datasets. Thus, the selection of a suitable clustering algorithm for a given application highly depends on the objective of the underlying task and, obviously, the nature of the data.

Feature Extraction

When considering natural data, data mining or machine learning techniques often face the problem that the data is not only large in terms of number of observations, but also that the number of features is large, i.e., the number of dimensions that a single observation consists of. Recalling the example on social data, the features of a user can be miscellaneous. For instance, they

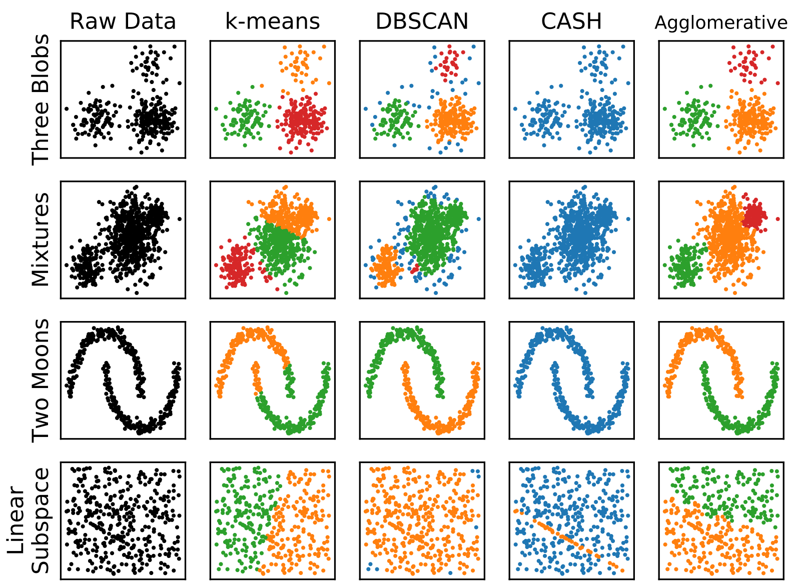


Figure 1.1: Clustering results for four clustering algorithms on four different datasets. The colors denote group membership as identified by the corresponding algorithm. Blue colored dots are considered as noise.

may consist of numerical features like age, height, years of education, salary, and so forth, but also of categorical features, e.g., mother tongue, music she likes or the user's profession. Categorical features are especially challenging as they typically appear in string format and usually cannot be used as is by most algorithms since they require numerical values. Obviously, simply transforming such features into nominal values, i.e., assigning an unique integer value to each category, is not an option because the resulting values are still unrelated to each other. Oftentimes, categorical data are therefore represented as one-hot encoded vectors, i.e., potentially huge zero vectors where each position in the vector represents one category and the value in the position of the corresponding category is set to one. However, this may lead to infeasible high-dimensional data representations which in turn is highly problematic for lots of data mining and machine learning algorithms as they typically tend to suffer from the curse of dimensionality, overfitting and/or high computational costs.

To overcome these issues, *feature extraction* is a fundamental and often indispensable task. In general, feature extraction describes the process of combining input features such that the amount of features is reduced by simultaneously keeping the loss of information contained in the data minimal.[3] Particularly, feature extraction typically reduces redundancy in the data, e.g., by combining correlated features into latent variables. In the above given example for instance the features age, years of education and salary are likely to correlate and hence contain some redundancy. In opposite to feature engineering, which describes the process of constructing hand-crafted features that may improve the outcome of an application task, feature extraction approaches usually are automated and learn feature combinations from the data in (un)supervised manner. For numerical vector data, one commonly used method is the principal component analysis, which aims at minimizing correlations within the input data $X \subseteq \mathbb{R}^d$ by transforming the data into some lower dimensional vector space $\mathbb{R}^p$, with $p < d$, where the basis is defined by the eigenvectors associated with the p largest eigenvalues. When dealing with complex, unstructured data like text or images, another common approach to feature extraction is the bag of words model where words in a text are represented by their frequency, or some related numerical value, disregarding the information about a word's location within a sentence or grammatical details. Analogously, it is possible to extract such features in other domains, too. In image processing for example, images can be represented as bag of words by defining specific image features, e.g., the frequency of pixel colors, as "words" [237]. Lastly, a large group of feature extraction methods are based on neural learning. Essentially, every pair of consecutive layers of any neural network, or more specifically every trained weight matrix between two consecutive layers, with the second layer consisting of less neurons than the first, that is not part of the classifier, resp. regressor, can be considered a feature extractor. These methods of feature extraction are typically referred to as *feature learning* or *representation learning*.

Neural Learning

Neural networks have proven to be particularly powerful in lots of applications and achieved state-of-the-art results in many of them. As neural learning is used oftentimes throughout this thesis (unsupervised and supervised learning for both presented methods and their evaluation), the basic concepts are briefly reviewed in the following. Figure 1.2 depicts a sample architecture of the simplest form of neural networks, i.e., a feed-forward neural network. This network consists of three layers, i.e., an input layer that

[3]This is not to be confused with *feature selection* where the dimensionality is reduced by filtering out irrelevant features.

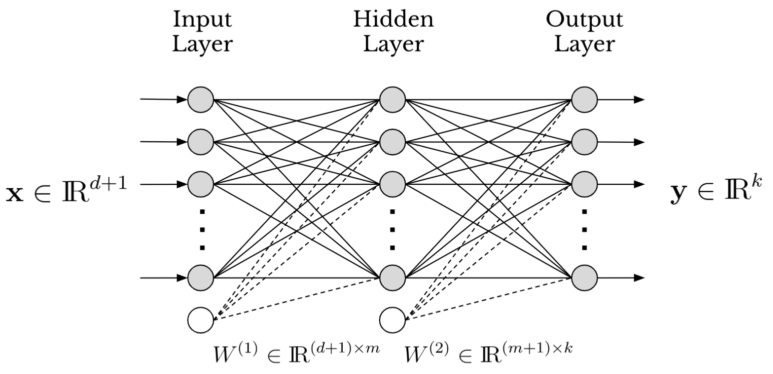

Input Layer
Hidden Layer
Output Layer
$\mathbf{x} \in \mathbb{R}^{d+1}$
$\mathbf{y} \in \mathbb{R}^{k}$
$W^{(1)} \in \mathbb{R}^{(d+1)\times m}$
$W^{(2)} \in \mathbb{R}^{(m+1)\times k}$

Figure 1.2: A sample feed-forward neural network with an input layer, one hidden layer and an output layer. Gray nodes represent variables, solid links denote weight parameters and the dashed links coming from the white nodes denote bias parameters.

consists of d units (or artificial neurons, resp., neurons) and one additional neuron for the bias parameter, an hidden layer that consists of m neurons and one additional bias neuron, and an output layer consisting of k neurons. Note that the number of neurons in the output layer depend on the problem that shall be solved. For regression tasks, which aim at inferring one continuous value for some given instance, k should equal to 1, whereas for classification tasks, where the goal is to assign input instances to one or many classes, k corresponds to the total number of classes. The hidden layer is densely connected to the input layer, which means that each neuron of the input layer is connected to each neuron of the hidden layer. In total there are $d + 1 \times m$ connections between the input layer and the hidden layer with each connection corresponding to an adjustable weight parameter. The collection of weight parameters is denoted as weight matrix $W^{(1)} \in \mathbb{R}^{(d+1)\times m}$. The output h_j of a single neuron j in the hidden layer is

$$h_j = f(\sum_{i=0}^{d} x_i w_{ji}^{(1)}) = f(\mathbf{x}^T \mathbf{w}_j^{(1)}),$$

with $\mathbf{w}_j^{(1)} \in \mathbb{R}^{d+1}$ denoting the j-th column vector of weight matrix $W^{(1)}$, w_{ji} being the i-th entry in $\mathbf{w}_j$, and $\mathbf{x} \in \mathbb{R}^{d+1}$ being the column vector of d input signals and an additional input x_0 being fixed to 1 since w_{j0} represents the bias. Function $f(\cdot)$ is the so-called activation function, which is in case of classification some differentiable and non-linear mapping $f : \mathbb{R} \to [0, 1]$, resp. $f : \mathbb{R} \to [-1, 1]$, or in case of regression the identity function. Analogously, passing one fixed bias input $h_0 = 1$ along with the m outputs of the hidden layer's neurons in a vector $\mathbf{h} \in \mathbb{R}^{m+1}$ as input into the subsequent output layer, the outputs of the single output layer's neurons can be computed by linearly combining the corresponding inputs as follows,

$$y_k = \sigma(\sum_{j=0}^{m} h_j w_{kj}^{(2)}) = \sigma(\mathbf{h}^T \mathbf{w}_k^{(2)}).$$

This time, $\mathbf{w}_k^{(2)} \in \mathbb{R}^{m+1}$ denotes the k-th column vector of the second weight matrix $W^{(2)}$, and $w_{kj}^{(2)}$ is the j-th value of $\mathbf{w}_k^{(2)}$. Due to the output layer being densely connected to the hidden layer, each output neuron's quantity is the result of a linear combination of all input values passed to some activation function $\sigma(\cdot)$. Differently to the hidden layer, the activation function for the output layer must be chosen more carefully depending on the underlying problem that shall be solved. Again, in case of regression problems, the activation function is the identity function. In case of classification problems, where input instances are assigned to discrete categories, one typically has to distinguish between *multiclass* and *multilabel* problems. In multiclass problems, each instance is assigned to exactly one of more than two classes. The classification problem where input instances are assigned to one of two classes is called binary classification problem. Contrary, in multilabel problems, a single instance can be assigned to multiple classes simultaneously. Formally, both multiclass and multilabel problems require some input vector $\mathbf{x}$ and retrieve some output vector $\mathbf{y}$ whose size corresponds to the number of classes k. In terms of neural networks, each of the k output neurons corresponds to one of the k classes, resp. to one of the k positions in $\mathbf{y}$. However, as the objective for multiclass problems is to maximize the class probability for one particular class in dependence on the probabilities of the other classes, the arbitrary, real valued input signals (called logits) arriving at the neurons of the output layer need to be transformed into probabilities that sum to one properly. Therefore, the so-called softmax function, i.e.,

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{l=1}^{k} e^{z_l}},$$

with any z_i with $i \in \{1, ..., k\}$ denoting the logit passed into neuron i and e being the standard exponential function, is employed as activation function. In fact, this normalized exponential function can be interpreted as a smoothed version of the argmax function. On the other hand, for multilabel classification, the probabilities for each class shall be independent from each other. Therefore it is desirable to get probabilities that denote the class membership for each class independently rather than getting probability values in dependence on the other classes. This means that the probabilities do not have to sum up to one in the multilabel case. In practice, the logistic sigmoid function, i.e.,

$$\sigma(z_j) = \frac{1}{1 + e^{z_j}},$$

with z_j denoting the logit passed into neuron j and again, e denoting the standard exponential function, has proven to be particularly useful as activation function for the output layer in neural network-based multilabel classification tasks. For binary classifications one typically uses either a single output alongside with the logistic sigmoid activation or two outputs with softmax activations.

Putting all together, a single forward pass aiming at predicting the outcome for neuron k in the output layer of the neural network depicted in Figure 1.2 can be written as

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma(\sum_{j=0}^{m} f(\sum_{i=0}^{d} x_i w_{ji}^{(1)}) w_{kj}^{(2)}),$$

with $\mathbf{w}$ denoting the group of weight and bias parameters, and $\mathbf{x}$ being the input vector for the neural network.

Beside the architecture of the neural network, the network training, which is necessary to adjust the weight parameters such that finally an accurate prediction model is retrieved, requires a set X of training instances, and a corresponding set Y of annotations, respectively ground truth labels, or short labels. Note that for classification tasks the labels are typically represented as k–dimensional binary vectors where k corresponds to the number of classes. A label vector has an 1 entry at position i if the corresponding instance is of class i, and a 0 entry otherwise. The general goal of the network training is to optimize the network's weight parameters such that the resulting model can map instances $\mathbf{x}_i \in X$ from the input distribution to the corresponding output vector $\mathbf{y}_i \in Y$. Therefore, training instances are given as input to the network, the input signals are transformed with respect to the weight parameters, and finally the output vector $\hat{\mathbf{y}}_i$ of the network is compared to the ground truth label $\mathbf{y}_i$ of the corresponding training instance $\mathbf{x}_i$. The latter step intents to retrieve a quantitative score on how good the prediction of the model actually is. Obviously, the objective of the network's training phase is to minimize the discrepancy between the output of the network $\hat{\mathbf{y}}_i$ and the actually desired output $\mathbf{y}_i$. For being able to quantify that discrepancy, or error, the neural network requires a loss function which typically depends on the activation function used for the last layer of the network. Considering the multiclass classification problem for which the softmax activation function is employed, the typical choice of loss function is the multiclass cross-entropy error function:

$$E(\mathbf{w}) = \sum_{i=1}^{N} \sum_{j=1}^{k} (\hat{y}_{ij} \cdot ln(y_j(x_i, \mathbf{w})),$$

with N denoting the number of input variables for training, k again being the number of output neurons, $\hat{y}_{ij}$ being the ground truth label for neuron j in the output layer for training instance i and $\mathbf{w}$ denoting the weight parameters of the model. When using the sigmoid activation function, as for multilabel classification tasks, a suitable choice for the loss function is the cross-entropy error function:

$$E(\mathbf{w}) = \sum_{i=1}^{N} \sum_{j=1}^{k} (\hat{y}_{ij} \cdot ln(y_{ij}) + (1 - \hat{y}_{ij}) \cdot ln(1 - y_{ij})).$$

Note that the previously presented loss functions are commonly used options and that they are used throughout the thesis unless stated differently. However, those loss functions are not exclusive for the corresponding tasks. Other loss function may be suitable, too.

Given the multi-variable loss function $E(\mathbf{w})$, the goal of the network training is to minimize the loss function such that the prediction errors are as small as possible. In this context, the phrase “as small as possible" refers to the fact that it is not essentially necessary to find a global optimum, respectively minimum in this case. Precisely, the optimization problem is relaxed in the sense that finding a local minimum for the loss function is sufficient for the network training. A broadly used optimization technique to find local minimums is the iterative gradient descent method. The general idea is as follows: given a differentiable multi-variable function $F(x)$, $F : \mathbb{R}^d \to \mathbb{R}$, with $d \geq 1$ and some input $\mathbf{x} = (x_0, x_1, ..., x_{d-1})$ such that $F(\mathbf{x}) = y$, then y approaches its (local) minimum fastest if $\mathbf{x}$ is changed towards the direction of the negative gradient[4] of $F(x)$ at position $\mathbf{x}$. By using a sufficiently small weighting factor $\eta \in \mathbb{R}^+$ for the gradient, it holds that

$$\mathbf{x}_{n+1} \leq \mathbf{x}_n - \eta \nabla F(\mathbf{x}_n).$$

Note that $\nabla F(\mathbf{x}_n)$ denotes the gradient of $F(x)$ at position $\mathbf{x}_n$ and η is the weighting factor which we refer to as learning rate in the following. The learning rate is typically decreased with every iteration which may slow down

[4]The gradient is a generalization of the derivative in the sense that it computes the slope of the tangent at a specific point of a multi-variable function.

the convergence of the gradient descent but has proven to be effective in avoiding oscillation around the minimum.

Considering that the number of adjustable weight parameters within a neural network is usually quite large, e.g., up to several millions for state-of-the-art convolutional networks [286], the calculation of the gradients, which can mathematically be seen as an aggregation over the partial derivatives of some multi-variable function $F(x)$ [5], must be done efficiently to enable the training of sufficiently good models within a feasible amount of time. The common way to calculate the gradients of error functions in a neural network is to use the error backpropagation algorithm. Recalling that input signals are transformed into output signals during a single forward pass, and that the output signals are subsequently used to determine some error by comparing them with the desired output signals, the error backpropagation is finally used to propagate the error backwards such that the weight parameters can be adjusted properly. Formally, the error which is propagated backwards to some hidden unit j after doing a forward pass for some input vector $\mathbf{x}_n$ is given as

$$\frac{\partial E(\mathbf{x}_n)}{\partial a_j} = \sum_k^K \frac{\partial E(\mathbf{x}_n)}{\partial a_k}\frac{\partial a_k}{\partial a_j} = h'(a_j)\sum_k^K w_{kj}\frac{\partial E(\mathbf{x}_n)}{\partial a_k},$$

with $E(\cdot)$ being the loss function, a_j denoting the value arriving at unit j, $h'(\cdot)$ denoting the derivative of the activation function used for the hidden layer, K being the number of units in the subsequent layer and w_{kj} being the weight parameters on the links connecting unit j with the units of the subsequent layer. The basic intuition behind error backpropagation can therefore be interpreted as a message passing technique where partial derivatives are passed backwards through the entire neural network by making use of the chain rule for partial derivates.

1.3 Contribution and Thesis Structure

The research presented in this thesis has been conceptualized, implemented and evaluated by the author of this thesis in cooperation with his supervisor Prof. Dr. Peer Kröger, the researchers and students from the Department for Database Systems and Data Mining at the LMU Munich as well as partly with researchers at the UC Berkeley and the George Mason University. Some of the content has previously been published at peer-reviewed conferences in

[5] E.g., $\nabla F(a,b,c) = \frac{\partial F}{\partial a}\mathbf{e_a} + \frac{\partial F}{\partial b}\mathbf{e_b} + \frac{\partial F}{\partial c}\mathbf{e_c}$, with $\mathbf{e_a}, \mathbf{e_b}$ and $\mathbf{e_c}$ denoting the unit vectors in directions a, b and c.

the field of Data Mining and Machine Learning. However, it is noteworthy that the published content has been revised and restructured in order to have an improved reading flow. In the following, the contribution of the thesis' author to the works presented in this thesis, as well as the thesis structure are summarized.

The content of this thesis is presented in two parts. The first part addresses various problems for which we present algorithms that are located in the field data mining. The second part is on representation learning on graphs where we present different methods that rely on neural learning techniques to learn models capable of embedding graph vertices or entire graph structures within continuous vector spaces.

Part I: Mining Social Data. The works presented in the first part of the thesis include solutions for oriented subspace clustering on data streams, an approach for generating and classifying user mobility profiles with the goal of user identification and a short work that has been published as vision paper presenting a proof-of-concept on how to use text mining methods with unsupervised, hierarchical clustering to determine socio-textual maps from social media data. The project on the PCA based correlation clustering algorithm for streaming data is a collaborative work with Thomas Hubauer from Siemens AG and Prof. Peer Kröger [48, 49]. The conceptualization, parts of the implementation as well as the evaluation was done by the author in cooperation with Peer Kröger. A preliminary proof of concept has been implemented by Matthias Schreiber during his master thesis. The student was supervised by the author. The domain knowledge for the use case was contributed by Thomas Hubauer, who also supported the process for the patent application. The concept for the correlation clustering approach on streaming data that relies on Hough transformation was developed in cooperation with Daniyal Kazempour and Peer Kröger[6]. It has initially been implemented and evaluated in the master thesis of Felix Mathy, who was advised by the author together with Daniyal Kazempour. The project on user identification, which was a joint work from researchers of our group and Assistant Professor Andreas Züfle and his graduate student Erik Seglem from George Mason University, evolved from a former project which investigated privacy issues in geo-tagged social data [228]. The author's contribution include the implementation of a proof of concept, ideas on how to represent trajectory data such that trajectories are suitable as input for classification models and the preparation of the manuscript. The work on socio textual mapping was developed with Michael Weiler, Andreas Züfle and Tobias Em-

[6]This work is planned to be submitted soon.

rich [262]. The author of this thesis contributed to discussions, supported Michael Weiler for the implementation and also supported the group to write the manuscript.

Part II: Representation Learning on Graphs. In the beginning of the second part of the thesis, we present solutions on unsupervised learning of node embeddings by realizing different inductive biases. Precisely, we present two methods, i.e., one for learning homophily based node embeddings and one for learning structure based node embeddings. The latter embeddings are further used to present an unsupervised method on how to generate useful representations for entire graph structures. Furthermore we present a semi-supervised method for learning node embeddings as well as an ongoing work on a supervised method that learns node matchings for the purpose of knowledge fusion at the example of map fusion tasks. The work on homophily based node embeddings was developed during a research stay at UC Berkeley [91, 92]. The ideas were developed, implemented and evaluated by the author together with his colleague Evgeniy Faerman. The ideas have further been discussed with Kimon Fountoulakis and Prof. Michael Mahoney from UC Berkeley. The work on structure based node embeddings was conceptualized together with Evgeniy Faerman and supported by Julian Busch [47]. A proof-of-concept was implemented by the author and evaluated together with Julian Busch. For further studies on the problem and the project on graph representations, Adina Klink initially implemented and evaluated a couple of concepts for her bachelor thesis which was supervised by the author, Julian Busch and Evgeniy Faerman [46]. The work on semi-supervised learning on graphs using local label distribution was developed and implemented together with Evgeniy Faerman [89, 90]. Julian Busch supported writing the manuscript. Finally, the project on Map Fusion was conceptualized together with Evgeniy Faerman. The models have been implemented and evaluated by Otto Voggenreiter in his master thesis that was supervised by the author, Evgeniy Faerman and Dr. Tobias Emrich (Harman/Becker Automotive Systems GmbH).

Part I

Mining Social Data

Chapter 2

Introduction

For decades, Knowledge Discovery in Databases (KDD) is one of the most important research areas in the field of data processing. It generally aims at providing knowledge from huge masses of data that are due to the vast amount hardly comprehensible and understandable by humans. The KDD process [94, 110], as depicted in Figure 2.1, summarizes the basic, and necessary steps to get from the raw data to the point where humans gain knowledge from such huge amounts of the generated data. Starting with the raw data,

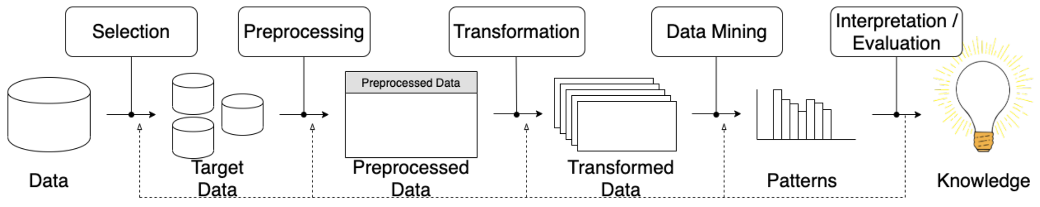


Figure 2.1: Illustration of the KDD Process.

the first step is to select the target data from the database such that the selected data comprises task-relevant records. Given the target data, the next step includes data cleaning and preprocessing, e.g., missing data handling, data corrections or noise removal. Third is data transformation: this step aims at representing the data appropriately for the given goal or subsequent method, and could include tasks like feature reduction, data discretization or simple transformation steps like representing string values in vector format or numerically. The fourth step, denoted as *Data Mining*, describes the application of mathematical, respectively visualization, procedures to reveal yet unknown patterns within the data. Finally, these patterns are interpreted and evaluated by experts to generate useful knowledge during the last step. Note that the entire process is iterative and each step can be refined individually after evaluating the outcome of the previous iteration.

The first part of this thesis presents methodologies that are positioned in the *Data Mining* step of the KDD process and have been developed during the thesis work. According to Decker et al. [78], Data Mining is defined as "*a problem-solving methodology that finds a logical or mathematical description, eventually of a complex nature, of patterns and regularities in a set of data*". Stemming from the fact that data mining techniques generally consider the data generating mechanism as unknown [54] and hence do not leverage any prior knowledge, they basically differ from statistical methods in that they generate hypotheses automatically rather than checking pre-defined hypotheses. Also, in contrast to classical approaches in the field of machine learning, e.g., regression or classification models, data mining methods generally aim at identifying yet unknown patterns within the data in an unsupervised fashion, while those "classical machine learning approaches" learn supervised, i.e., with the goal to learn models that can recognize previously trained patterns within data. Due to the "*growing consensus that data mining can bring real value*" [65] and the fact that this "*has led to an explosion in demand for novel data mining technologies*" [65], the development of suitable data mining algorithms is a central part in the field of artificial intelligence and especially useful for machine learning tasks where no (or very few) prior knowledge, pre-defined patterns or ground truth labels are present.

In this part, we first focus on methods for correlation clustering in data streams, i.e., methods that aim at detecting arbitrarily oriented subspaces within high-dimensional data by simultaneously coping with data that is generated, respectively changed by external influences, with high velocity. Precisely, we present two approaches: one is biased towards finding local correlation clusters and is based on the Principal Component Analysis (PCA) to identify relevant subspaces, and the other approach is based on Hough transformations and is designed to detect arbitrarily oriented subspace clusters on a global level. Such correlation clustering methods for streaming data have high potentials in many applications dealing with high-dimensional data. For instance, these include sensor data analytics where potential machine failure states may be identified when tracking correlations among specific subsets of sensor measurements, or social data analytics where one might be interested in user groups that may cluster when considering a certain subset of attributes, e.g., textual overlap within microblog posts. Next, we present an analysis on identifying users of a social platform by reconstructing spatio-temporal user profiles from geo-tagged posts. Our findings show that not only users who post in incognito mode can quickly be identified by only considering their spatio-temporal traces, but also that even fairly simple data mining methods are sufficient to link user profiles from different social media platforms. Finally, this part concludes by presenting a preliminary work on socio-textual mapping, where blog posts are used to construct temporally dynamic maps that reveal regions where people follow the same trends, respectively discuss the same topics.

Chapter 3

Related Work

3.1 Subspace Clustering

In the literature, the term subspace clustering has been used ambiguously to describe different classes of clustering problems that focus on finding clusters in high-dimensional data. However, as the corresponding work presented in this part of the thesis mainly concentrates on a particular (and generalized) problem in the field of subspace clustering, i.e., the detection of arbitrarily oriented subspace clusters, we briefly review the standard vocabulary presented in [148] to avoid potential confusions before giving an overview of the previously published works in that area.

In general, all subspace clustering algorithms have in common that they aim at grouping high-dimensional objects by considering only a subset of the entire feature space, i.e., the so-called *relevant* features, for the clustering process. Formally, in terms of subspace clustering, a clustering is described as a set of pairs (X, Y), with X being a subset of data objects and Y being a subset of features. When projecting the objects in X onto the features of Y, the objects in X are supposed to be spatially close to each other with respect to some distance measure, and spatially distant to each other when consider other features than those in Y. Generally, one can distinguish three classes of subspace clustering algorithms. In the first class, consisting of the axis-parallel subspace algorithms, features are considered as-is (which means that they remain unchanged) and the goal is to find certain combinations of features such that clusters of objects can be identified within the subspace spanned by only those relevant features. Since the features are not manipulated further, the resulting subspaces are restricted to be axis-parallel. Obviously, this is already a challenging problem since the number of potential subspaces, which corresponds to the complexity of the search space, is in

$O(2^d)$. In contrast, subspace clustering algorithms of the second class, i.e., the oriented subspace clustering algorithms, or correlation clustering algorithms, are designed to solve the even harder problem of detecting arbitrarily oriented subspace clusters within the entire feature space. In particular, this means that relevant features are not considered unchanged but can somehow be combined such that the resulting combination serves as a part of the base forming the arbitrarily oriented subspace. As a consequence of relaxing the restriction of subspaces to be axis-parallel, the search space complexity becomes unbounded and the number of such arbitrarily oriented subspaces is infinite. The third class of subspace clustering algorithms are the pattern-based approaches. These approaches work on the data matrix directly and are inherently different to those of the first two classes because they treat the data objects and the features interchangeably rather than defining the subspace first (w.r.t. the features) and then performing the actual clustering within the subspace (w.r.t. some distance measure). The idea behind the pattern-based approaches is to select subsets of features and objects simultaneously such that all of the selected objects follow a similar pattern when considering the selected features. In contrast to the algorithms of the first two classes, those patterns do not necessarily have to be based on spatial intuitions. Please note that the pattern-based approaches are just mentioned for the sake of completeness but left out when surveying the state-of-the-art since those approaches are out of scope of the thesis. The interested reader is referred to the survey article published in [243].

Beside the distinction of the different classes of subspace clustering algorithms, the literature also distinguishes between different problems that shall be solved with the subspace clustering algorithm. In fact there are two main classes, i.e., the problem where each data object shall be assigned to exactly one cluster, respectively k clusters, and the problem where each data object may be assigned to various clusters in different subspaces. The algorithms in the first category are called projected clustering algorithms, respectively "soft" projected clustering, and are characterized by aiming at finding the projection, respectively the k projections, where the set of data objects clusters best. The second category is composed of the subspace clustering algorithms[1] which aim at finding all clusters in all subspaces. In particular, this means that a single data object may appear in multiple clusters. It is noteworthy that there have also been hybrid algorithms proposed which are typically designed to detect overlapping clusters but usually do not search

[1]Note that the term subspace clustering algorithm is still overloaded as it is used to describe the algorithms of this specific category as well as the entire family of algorithms designed for clustering high-dimensional data. However, wherever used, the meaning should become clear from the context.

for all clusters in all subspaces.

Axis-Parallel Subspace Clustering

An early algorithm in the field of projected subspace clustering is *PROCLUS* and was presented in [16]. *PROCLUS* is an iterative approach that starts by selecting k_0 initial medoids as cluster centers and subsequently refines the clustering by assigning data objects to the currently considered k, with $k < k_0$, cluster centers with the objective to minimize the standard deviation of the distances along each dimension. After deriving the subspaces, the set of medoids is evaluated against certain thresholds and eventually replaced by *better* medoids from the initial set. The algorithm finally retrieves k clusters associated with the corresponding subspaces. This approach was further refined in [265]. *SSPC* [273] presents a semi-supervised k-medoid based variant for discovering low-dimensional projected subspace clusters within high-dimensional data. In particular, the presented method allows to make recourse to already labeled data instances for selecting the initial seed objects. The method published in [167], named *CLTree*, follows a different approach which relies on training decision tree to extract clusters and subspaces. The key idea is to use artificial labels on the data points and artificial uniformly distributed noise object with another label, and hence transforming the clustering problem into a classification problem. By training a decision tree they separate dense areas from sparse areas to finally detect clusters within the data space. The *PreDeCon* algorithm [41] is a density-based approach that first determines the preferred subspaces of the data objects by considering the variances along each dimension within certain neighborhoods, and subsequently uses a weighted Euclidean distance measure in combination with an adopted variant of the *DBSCAN* algorithm [87] to identify density-connected clusters within the subspaces. Somewhat similar, the *COSA* algorithm [96] also relies on subspace preferences derived from local neighborhoods, but in contrast to *PreDeCon* assigns continuous weights to the preferred dimensions (rather than discrete values in $\{1, \kappa\}$, with $\kappa \gg 1$), and finally retrieves a similarity matrix which can be used for subsequent clustering procedures. Note that weighting the dimension with continuous values results in a soft projected clustering here. In [81], the authors propose a k-means based scheme to detect projected subspace clusters. The general idea behind this approach is to weight (again, continuous weights) each dimension such that irrelevant (with respect to the corresponding subspace cluster) dimensions get weights close to zero. A very similar approach that enables the user to specify specific constraints can be found in [68].

To the best of our knowledge, the *CLIQUE* algorithm [20] is the first grid-based algorithm that is able to detect subspace clusters where data objects may be assigned to multiple clusters in an arbitrary number of subspaces. By partitioning the data space into an equi-sized grid, the algorithm identifies dense regions within low dimensional subspaces and subsequently finds higher dimensional subspaces in a bottom-up fashion. The idea of using a grid-based approach has been borrowed and algorithmically refined in several works, e.g., for the *ENCLUS* algorithm [67], the *MAFIA* algorithm [99], or the *nCluster* algorithm [168]. The latest variant, i.e., the *MaxnCluster* algorithm [169], uses a sliding window approach to detect maximal δ-nClusters (i.e., subspace clusters that consist of a maximal attribute and object set) that might be split into several smaller clusters when using a pure grid-based approach. *SUBCLU* [137] tackles the problem of subspace clustering by employing *DBSCAN* within lower dimensional subspaces. The density-based approach has been improved in [23, 24] by the *DUSC* method that uses a density measure which adapts to the subspace dimensionality rather than relying on a global density threshold. Another density-based method having the objective to eliminate the global threshold value and therefore aiming at determining individual subspace cluster thresholds adaptively is presented in [70]. In [105], the authors propose a subspace clustering procedure that is able to deal with uncertainties within the data.

The *DOC* algorithm [215] is a greedy Monte Carlo algorithm which computes projected subspace clusters (but not all clusters in all subspaces and hence it is a hybrid method) by determining dense subregions within a grid-like partitioning of the data space. Similarly, but in a deterministic fashion, *MINECLUS* [274, 275] aims at finding projected subspace clusters by transforming the problem into a frequent pattern mining problem. Yip et al. [272] present a top-down algorithm, named *HARP*, that uses a hierarchical clustering scheme to iteratively decrease the number of features per subspace cluster. Another hierarchical approach is *DiSH* [4], which is able to identify subspace clusters that are embedded within higher dimensional subspace clusters. The *FIRES* algorithm [146] follows a bottom-up strategy by first detecting one-dimensional subspace clusters which are later, with respect to a similarity function that relies on the overlap of clusters, merged to form higher-dimensional subspace clusters. *P3C* [184, 185] follows a somewhat similar strategy. The authors propose to mine maximal-dimensional subspace cluster approximations in a bottom-up fashion and subsequently refine them with an *EM* based clustering mechanism. Another subspace clustering method that explicitly guides the clustering procedure towards considering only the most relevant subspaces (the idea is based on the findings presented in [187]) which makes the algorithm highly scalable was presented in [188].

Correlation Clustering

The family of oriented subspace clustering, or correlation clustering, algorithms that have been developed in the community of data mining mainly can be categorized into two classes: approaches that rely on the Principal Component Analysis (PCA), and approaches that rely on parameter space transformations. Algorithms of either class are designed to detect arbitrarily oriented subspace clusters within the full dimensional space.

The class of PCA-based correlation clustering algorithms have in common that they use PCA for identifying low dimensional subspaces defined by inter-attribute correlations and subsequently project the considered data objects onto the relevant principal components to perform data object clustering. In general, a wide variety of such algorithms have been presented in the past. The first algorithm that was presented in this field of subspace clustering algorithms is the *ORCLUS* algorithm [17]. *ORCLUS* is an iterative k-means based algorithm which starts by assigning the data objects to k_0 initially chosen cluster seeds. In each iteration data objects are projected into the corresponding eigensystem (derived from the yet assigned data objects). A data object is newly assigned to the cluster in whose eigensystem the object is closest to the cluster centroid, with the distance being calculated with respect to the weak eigenvectors (i.e., those eigenvectors whose eigenvalues are small). After the assignment step, the closest pairs of clusters (again, the distance is calculated with respect to the projection onto the weak eigenvectors) are merged. The entire procedure is repeated until the previously user-defined number of $k < k_0$ clusters is reached. The *4C* algorithm [42] is another PCA-based method, but, in contrast to *ORCLUS*, uses density criteria for deriving the clustering rather than following the partitioning paradigm. Precisely, *4C* uses ϵ-neighborhoods (in Euclidean space) for deriving the covariance matrix which is then decomposed into the eigenvector and eigenvalue matrices. After replacing the small eigenvalues with large values, the manipulated eigenvalue matrix of a given data object is used to compute projected distances to other data base objects which in turn are used for expanding the density-based clusters. To improve this idea in terms of computational efficiency, Achtert et al. proposed *COPAC* [6] that partitions the data set into sets of points that exhibit the same local subspace dimensionality. The main advantage is the reduced complexity for detecting density-connected clusters within the subspaces. Another improvement is that *COPAC* determines the similarity matrices based on k-neighborhoods instead of ϵ-neighborhoods. The intuition behind this is the avoidance of sparsity issues that might occur when relying on ϵ-range queries in the high-dimensional space. Furthermore, the *ERiC* algorithm [5] was proposed to further improve the outcome of the clustering procedure of *COPAC* by introducing another processing step that aims at building a hierarchy of correlation clusters. This enables the procedure to retrieve arbitrarily oriented subspace clusters that are nested within higher dimensional subspace clusters. A similar approach that is also able to reveal hierarchical structures is the *HiCO* algorithm [7]. Finally, the *CURLER* algorithm [255] is also based on PCA and designed to detect non-linear subspace clusters.

The other group of correlation clustering methods follows another base mechanism. Instead of making use of the PCA, these approaches use parameter space transformations, i.e., the so-called Hough transformation [122, 83]. The original idea behind Hough transform is the transformation of data points from the data space into the parameter space, with the parameter space being defined by the distance r between the origin and the closest (wrt the origin) point of a straight line, and the angle θ between the x-axis and the connection between the origin and that closest point. In case of transforming a single data point, all straight lines going through this data point are considered. This results in single data points being represented as sinusoidal curves, resp. $(d-1)$-dimensional hyperplanes for a data point $x \in \mathbb{R}^d$, in parameter space. The *CASH* algorithm [3] is the first method that makes use of this parameter space transformations for correlation clustering as it uses the fact that an intersection of sinusoidal curves in the parameter space corresponds to the data points laying on a straight line in the data space. Using this, *CASH* detects correlation clusters by finding dense regions in the parameter space. Therefore it employs a grid-based procedure that relies on recursive descent to identify interesting subspaces. Since the search strategy may be rather expensive, the *D-MASC* [138] algorithm accelerates the search for dense regions within the parameter space by applying mean shift on the data in data space, transforming the generated modes into the parameter space and using the resulting functions to prune the search space. The *HCC* algorithm, proposed in [139, 140], uses a variation of the Hough transform to detect non-linear subspace clusters. By transforming the objects in data space into a somewhat different parameter space, i.e., by using parameters that describe paraboloids rather than linear functions, the subsequent search for dense regions finally retrieves parabolic correlated subspace clusters.

Further Subspace Clustering Approaches

Another group of subspace clustering techniques which have primarily been developed within the vision community are spectral methods [73, 193] and neural learning techniques [85, 203, 205, 212, 204]. These methods basically differ in how they learn some similarity graph, or affinity matrix, to denote relationships within the input data. Subsequent spectral clustering on the derived affinity matrix finally aims at finding subspace clusters. Based on the intuition of the self-expressiveness property [85], some approaches relying on deep learning, e.g., [211, 135, 282], have been proposed recently. Note that these approaches are mentioned for the sake of completeness as they are somewhat related (though they all have been published within a different community) and have been proposed very recently. However they follow a completely different paradigm as the related works presented in this thesis, and hence are not discussed in depth.

3.2 Stream Clustering

Mining data streams generally has gained lots of attraction in recent years as more and more data is produced continuously by a plethora of different devices or mechanisms, e.g., mobile phones, sensors, or user behavior monitoring mechanisms. As this thesis partially focuses on oriented subspace clustering algorithms for high-throughput data, we briefly review established methods on how to deal with potentially unbounded sequences of data, i.e., data streams, in general. Then, related work in the field of data stream clustering, especially subspace clustering on data streams, is reviewed in detail.

In the broad majority of presented techniques, data stream clustering consists of an online and a separate offline step. While the online step, also called data abstraction step, tackles the main challenge that stream algorithms typically face, i.e., handling the data in a single scan properly, the offline phase usually aims at partitioning the data. Precisely, the online step processes the data stream by aggregating the yet seen data into summary structures such that the key statistics needed for the offline step are kept. One important aspect here is that more recent data is usually more important for the knowledge discovery process since obviously, in most applications, e.g., machine monitoring or market/trend analysis within data provided by social media platforms, stale data quickly looses relevance for the task at hand. This particularly holds in applications where the underlying data generating processes reveal non-stationary distributions and hence exhibit gradually appearing concept drifts, respectively suddenly appearing concept shifts. Note that the mechanism of "forgetting" stale data is called *aging*. To make the algorithmic procedure concentrating on up-to-date data, several distinct window approaches have been proposed. Given a data stream, the *landmark window model* picks an object of the stream as landmark element either depending on time or number of objects seen since the previous landmark element. The landmark elements are basically used to partition the data stream into disjoints data chunks. Typically, the landmark approach proceeds as follows: once a new landmark is reached, all objects that are kept in the current window are aggregated within some summary structure and subsequently can be discarded. Then, new incoming objects are stored until the next landmark appears. If this happens, again, relevant statistics of the current bunch are aggregated and the raw data objects are discarded, and so on. The next commonly used approach is the *sliding window model.* The idea is to load a certain amount of data objects into a data queue that employs the first in first out concept. Once the queue is filled, the summary structure keeping the relevant statistics from the data chunk is computed and kept in memory. As new objects arrive and added to the queue, the firstly inserted (and oldest) objects are removed, and the summary structure is updated accordingly. Another well-known technique is the *damped window model*, where each data object is associated with a weighting factor that degrades over time. This leads to the desired effect that the older objects become, the lower their weights become, which finally corresponds to the idea that more recent objects are considered more important. A widely used weighting function is the exponential decay

$$f(t) = 2^{-\lambda t},$$

with t denoting the current time stamp and $\lambda > 0$ being the decay rate. It holds that the lower the decay rate λ, the more importance is given to stale data, and vice versa.

For summarizing relevant statistics of data provided by streams, a common technique used in many of the related works is the so-called *(clustering) feature vector* (CF) data structure, originally presented in connection with the *BIRCH* algorithm [278]. The key characteristic of this data structure (and similar abstractions that follow the same objective) are its incrementality and additivity. Incrementality is the property that a feature vector can be updated by inserting new data objects; additivity means that two disjoint feature vectors can be merged into a new feature vector by summing up the single components. In case of *BIRCH*, a CF vector is composed of the number of data objects, the linear sum of the data objects and the squared sum of data objects. Importantly, these components fulfill the required properties incrementality and additivity and additionally allow the computation of the centroid, the radius and the diameter of the corresponding cluster. Those measures in turn can be used in the offline step to define the final clustering model. The *Scalable k-means* algorithm [53] makes use of the idea of CF vectors by using those aggregation structures to compress the statistics of data objects that are unlikely to change their cluster membership. This way, they enable an efficient way to compute *k-means* clustering on very large data sets. This idea is carried on in *Single-pass k-means* [93] by compressing fully disjoint data chunks, and keeping only relevant *k-means* statistics in memory. *CluStream* [14] extends the originally proposed form of clustering feature vectors by incorporating temporal information. The resulting data structure is called microcluster and the proposed algorithm employs a k-means based clustering on the microclusters. Another method that uses a slightly manipulated variant of the original structure of CFs can be found in [280]. In [60], the authors propose a density-based counterpart to *CluStream* which is called *DenStream*. By following the density-based clustering paradigm, they manage to get rid of the parameter k that requires to pre-define the number of microclusters by the user. Two similar approaches that mainly focus on the slightly related task of anytime clustering, i.e., stream clustering approaches that are capable of providing (possibly less accurate) results to the user whenever clustering results are requested, have been presented in [145, 114].

A somewhat less sophisticated, but in many applications sufficient, compression technique that is widely employed for stream clustering in general is to only keep track of the cluster representatives. The basic idea is to represent entire chunks of data solely in form of cluster representatives, e.g., cluster centroids. The *STREAM* algorithm [104] loads a user specified amount of data objects into memory and represents each chunk by $2k$ representatives employing a k-medoid like approach on the data. After having a certain number of representatives, these are further clustered into $2k$ representatives and the algorithm proceeds processing newly incoming data arriving from the stream. Likewise, the streaming variant of the *LocalSearch* algorithm [198] uses such a divide and conquer processing scheme in combination with k-medoids fully in-memory.

Further techniques to deal with the challenge of summarizing data streams are the coreset tree structure as proposed in [8] and the family of methods that use a grid-based summary strategy, e.g., as used in [61, 66]. However, the stream clustering algorithms presented in this thesis use variants of the former two methods and thus we refer to the survey in [235] for an overview of previously presented methods using the latter two summary techniques.

Subspace Clustering on Data Streams

As we focus on subspace clustering on data streams in this thesis, the following provides a somewhat more detailed review of related works tackling the problem of clustering high-dimensional data streams.

The first work able to cluster high-dimensional data streams properly was proposed in [15]. The *HPStream* algorithm is a projected subspace clustering method. Precisely, it is a k-means based approach that uses an adopted form of CF vectors to represent relevant cluster statistics. This data structure not only fulfills the additivity and incrementality properties, but also has the property of temporal multiplicity since the CF vectors, respectively fading cluster structures, underly an exponential decay over time. The key idea behind the stream clustering scheme is that each fading cluster structure is associated with a binary vector containing 1 entries for preferred dimensions, and 0 otherwise. If a new data object arrives from the data stream, *HPStream* uses these binary vectors to calculate the projected distance to the closest cluster structure for assigning the data object. To make the assignment more robust, the projected clustering is iteratively refined by refining the preferred dimensions for each cluster at least temporally for incoming data objects. Data objects are only absorbed if they lie within a specific radius of the chosen cluster. It is also noteworthy that the entire stream clustering process requires an initialization step to create an initial set of clusters.

The *IncPreDeCon* algorithm [147] is an incremental version of the density-based projected clustering algorithm *PreDeCon*. Although *IncPreDeCon* is designed to handle dynamic data in the sense that the projected clustering can be adopted incrementally, it does not support any form of aging and hence cannot deal with streaming data directly. Nonetheless, this algorithm already presents a solution to the incrementality property and is therefore a preliminary solution towards the density-based projected stream clustering algorithms *PreDeConStream* [115] and *HDDStream* [197]. Both *PreDeCon-Stream* and *HDDStream* have been developed simultaneously and rely on the basic ideas of the *PreDeCon*, respectively *IncPreDeCon*, algorithm. During the online phase, both methods aggregate incoming data objects within different microcluster structures, i.e., core, potential and outlier microcluster, and retrieve the final clustering by following (slightly different) variants of the density-based clustering scheme proposed in [41] during the offline phase.

The *SiblingTree* method presented [202] is a grid-based subspace clustering approach that aims at detecting all low-dimensional clusters in all subspaces. The idea is to start monitoring the data distributions in one-dimensional subspaces, maintaining the grid cells in a so-called sibling list, and splitting cells once they reach a certain density. Splitting the grid cells corresponds to considering higher dimensional subspaces, and by doing this iteratively, the algorithm actually builds up a tree like structure in order to support an efficient maintenance during the online phase.

A vaguely related work that mainly focuses on the task of feature selection in high-dimensional data streams can be found in [123]. By maintaining low-rank approximations of the observed data coming from the data stream the presented approach identifies the most interesting features. This can be interpreted as a global approach to projected subspace clustering, but however neither detects locally dense subspace clusters nor is able to retrieve precise information about different subsets of dimensions in which subspace clusters may exist.

3.3 User Identification

This section provides a survey of the state-of-the-art in mobility patterns of individuals, spatio-temporal user identification, user linkage, and spatial privacy as these are related to the work presented in Chapter 6.

Mobility Patterns

There are a variety of ways to study mobility patterns of individuals. One source of this data would be travel diaries such as in [264]. GPS devices have also been used [230], and more recently cell phone data has been used to study these patterns [132]. Social media, such as Twitter, can also be used. [239] are able to use it to harvest informations about groups. However there are some biases in the data, such as an individual only using social media when they are at certain locations or at certain times of the day it is still usable [125]. This can still provide frequently visited locations. Identifying these major locations provides a good basis for activity pattern analysis such as in [121].

Day to day variability in activity patterns have been shown in [111]. [29] establish that a single day is not sufficient enough to capture a persons regular activity pattern, but their lives do revolve around a number of major locations. Thus, they are not completely random, and do contain a pattern [112]. If data is collected only over a day or two, it can be thrown off by random activity, hence a longer period of collection may be necessary [100]. [120] establish that a coarser granularity is better at general representation of mobility patterns. [124] aggregate longer periods of time, in order to gather mode samples to offset the sparsity of Twitter data.

User Identification

A problem similar to the problem of trace based user-identification was considered in [77]. This work estimates the number of points needed to uniquely identify an individual trace. The dataset contained 15 months of data on 1.5M users in a small European country. Each time a user connected to a mobile phone tower to send or receive a call or text message, a tower location and time, with a resolution of one hour, was recorded. There are almost 6500 unique antennas in the dataset, and on average each user has 114 interactions per month. Among this dataset, they found that four randomly chosen points in a trace were enough to uniquely identify 95% of the trace, and two randomly chosen points were enough to identify 50% of the traces. However, the question whether a trace is unique, is different to the problem of user-identification tackled in this work.

The user-identification method in [77] assumes that a trace of the user to be identified is already in the database. Thus, a new trace, which has not been seen before, cannot be classified. Summarizing, the work in [77], aims at identifying individual traces, rather than individual users. Their work provided an initial framework to build this work on.

The work presented in [37] investigates the problem of how to prevent the identification of actual persons behind the users of location based services. Thus de-anonymizing the user. Therefore, the authors employ so-called location-based quasi identifiers, which are formed from historical spatio-temporal movement patterns that are gathered from location-based service requests as a potential privacy concern. However, the stated problem is slightly different from this work, as they make use of external sources to finally get the real names behind the pseudonames.

User Linkage

There are a variety of publications considering the problem of user linkage or more general record linkage. In the database community, record linkage generally aims at detecting duplicate records within one or several databases. Records describing the same entity may not share a common key or contain faulty attribute values, which makes the detection of such duplicates non-trivial. A survey on the proposed approaches can be found in [86].

Considering networks, record linkage is widely understood as user linkage and is stated as the problem of linking corresponding identities from different communities appearing within one or many networks [276]. It is specifically tailored to the requirements of user identification in heterogeneous data by considering co-occurrences adjusted with a stimulus signal. The stimulus signal is derived from locations with frequent co-occurrences and decays with increasing distance to a trajectory. The stimulus signal allows this method to weight important locations, which helps to distinguish two users with very similar trajectories.

An important area of user linkage is social networks where the user linking problem aims at connecting user profiles from different platforms that are used by the same persons. [170] differentiate between three types of user linkage across social networks: user-profile-based methods, which use information provided by the profiles to connect corresponding profiles [179], user-generated-content-based approaches, which analyze the content published by the users to link profiles [170] and user-behavior-model-based methods that generate models based on the (temporal) user behaviors and finally link user profiles based on the similarity of these models [172].

Most related to this approach is the recent work of [63]. In this work, the authors use various sources for data for the trajectories and propose a MapReduce-based framework called Automatic User Identification (AUI). They identify sample rate, temporal and spatial sparsity, and the fact that people with a close relationship provide similar trajectories as distinct features of the data. Sparsity of the data is corrected by using a long time frame. Signal Based Similarity (SIG) is introduced as a measurement of the similarity of two trajectories. In contrast to that approach, this work uses sparser trajectories. While the authors of [63] do consider sparse social media data, they accumulate these trajectories during a long time interval of at least multiple months. In this work, a long term mobility history of user is not assumed to be available. Instead, it aims at identifying users with the fewest observations possible.

Spatial Privacy

The predominantly used measurement for privacy is k-anonymity [242], which works with a closed world assumption and assures that, for each query that could be used to identify the identity of a user, at least $k-1$ other users are returned as possible results.

Common approaches to guarantee a defined degree of anonymity are suppression, obfuscation and generalization [113]. To achieve k-anonymity by suppression, every element that does not fit into an anonymity set is removed [58, 151]. For trajectories, suppression would require discarding observations in discriminative locations such as a user's home. While this method is effective, the use of suppression alone can lead to a significant loss of information. Perturbation is another method used to obfuscate the data [12]. The goal is to generate a synthetic dataset with the same properties of the original dataset using a generative model. For generalization, k-groups of users could simply be unified into a single entity.

This work does not try to maintain privacy of users, and can be seen as an adversary approach of trying to breach the privacy of users. A highly relevant future piece of work is to investigate how existing privacy preservation methods for trajectories can be employed to suppress, obfuscate and generalize trajectories to minimize the user identification accuracy of this solutions, while further minimizing the loss of information in the data.

A more refined version of k-anonymity is l-diversity, which addresses some shortcomings of k-anonymity [177], mainly where properties of the data are homogeneous and allow conclusions, which might violate the assured k-anonymity. Regarding trajectories, location l-diversity is required as introduced in [32]. As an enhancement of l-diversity, t-closeness [160] is used on datasets where the distribution of attribute values allows conclusions to identities.

These measurements are typically applied when medical records are published or in regards to Location Based Services (LBS), which require personalized location information. As LBS are usually working with GPS coordinates and trajectories, the raw data is similar to the information used in this work. But there is a difference in quality and frequency. LBS usually work with the assumption that a user is willingly providing their location as precise as possible and/or performing measurements of the location with a high frequency. While work has been done on interpolating real trajectories from purposefully obfuscated ones [190], the data used is limited to one service and focusing on the k of k-anonymity instead on user identification.

The work of [2] applies k-anonymity on spatio-temporal objects introducing the (k, δ)-anonymity. The trajectories of a user are extended by the uncertainty of the location measurement δ. The authors claim that a series of trajectories and locations can be modeled as a series of cylinders, or a tube. k-anonymity is granted when $k - 1$ additional elements of the set can fit into a tube. The proposed method uses outlier detection and other forms of suppression in combination with space transformation of a maximum of $\delta/2$ while δ defining the circumference of the tube remains unchanged. The paper proposes a heuristic that succeeds to find anonymity sets as the problem is NP-hard.

The notion of (k, δ)-anonymity is also discussed in [251]. The authors come to the conclusion that existing methods to create (k, δ)-anonymity as developed in [2] are not sufficient if $\delta > 0$. By defining every location in a spatio-temporal trajectory as a quasi-identifier and assuming that a potential adversary has knowledge about one sub trajectory they show that the probability to correctly identify a series of trajectories is larger than $1/k$ thus violating the k-anonymity. This work will show that it is indeed possible to identify users with high probability by only knowing a sub trajectory.

3.4 Text Clustering

Several techniques in the field of text clustering have been proposed recently. Due to presenting a work on determining spatial areas that differ in the current topics of social media blog posts in Chapter 7, we briefly review basic related techniques for text clustering in the following.

Particularly in the context of clustering text originating from social media platforms, feature selection methods are useful to discard words without informative content, i.e., words that appear very frequently. A common approach is referred to as document frequency-based selection, introduced by [175]. The idea is to weight those terms that appear more frequently with a higher value. To avoid prioritizing stop words like *the*, [238] extended this approach by using *inverse document frequencies*. However, TF-IDF weighting gives no information about the importance of a term which is often needed when considering further documents or blogs, e.g., for the purpose of clustering. [76] proposed a concept that uses an entropy measure to select those features that carry the most information. Since this method is quite inefficient for a large number of terms, the authors proposed to use sampling if the dataset is large. In [173], Liu et al. present a filter method based on the contribution of a term to the similarity of two documents. Let us note that the latter technique depends on the assumed similarity measure used for the clustering process. The aforementioned methods assign numeric values to the selected terms which can be used to get a vector space model of the corresponding document. Further transformation methods, like Latent Semantic Indexing [79] or non-negative matrix factorization [267], can be applied to reduce the dimensionality of such feature vectors to achieve better clustering results when using the retrieved latent-space vectors. Also, in the natural language processing community, lots of development effort has been spent in learning representations on different scales, e.g., word embeddings, sentence embeddings, or document embeddings. They all aim at representing natural language in meaningful vector representations which can be used for subsequent clustering tasks. An extensive survey on text mining algorithms can be found in [19].

Chapter 4

PCA-based Correlation Clustering on Data Streams

The work presented in this chapter has been published as the article *A Generic Framework for Correlation Clustering on Data Streams* in the Proceedings of the twelfth International Conference on Similarity Search and Application, 2019 [49]. A second version with focus on the application for rotating equipment is filed as US Patent US10339784B2 under the title *Method and system for monitoring sensor data of rotating equipment* [48]. The previously published patent applications are filed as US2017365155A1 and EP3258333A1.

4.1 Introduction

Due to the widely deployed data gathering devices, such as sensors, smartphones, tablets, etc., data stream analysis has gained more and more importance in the past few years. Especially the analysis of high-dimensional data streams has become an important challenge since such data is often extremely hard to analyze without using proper software solutions. A common approach to discover knowledge in high-dimensional data is *correlation clustering*. Correlation clustering generally reveals dependencies between different features and aims at reducing the feature space such that clusters can be clearly distinguished from other clusters by regarding different combinations of features.

Several subspace clustering methods have been proposed in the past but most of them are limited to static databases and the few algorithms that can deal with streaming data basically concentrate on detecting subspace clusters in projected, i.e., axis-parallel, subspaces. However, in many applications it might happen that complex relationships in form of correlations between different features appear. Finding such correlations can give a deep insight into the nature of data. Regarding a sensor controlled turbine for instance, a relatively simple correlation would be the relationship between the energy production and the speed of the rotation. Generally, having knowledge about hidden dependencies that occur during standard operation of a machine can be very useful in monitoring systems. Assuming that the correlation between rotational speed and produced energy of a turbine slowly or suddenly breaks could portend a failure scenario, e.g., a broken blade, and an alert can be initiated. Correlations between different features cannot be detected by projected clustering approaches, in general. Although there exist some methods to identify arbitrarily oriented correlation clusters within data, these techniques fail when considering streaming applications like monitoring systems. Further interesting applications for correlation clustering algorithms are recommendation systems, where the knowledge of customer groups with correlated affinities can be very helpful for companies when considering target marketing for instance. Especially online shops can profit from detecting such user groups and assigning an active user to such a group in real time since recommending proper products or ensuring a comfortable surfing experience on a self-adaptive website are key requirements for customer binding.

In this chapter, we present a generic framework for *PCA*-based correlation clustering called CorrStream which is able to cope with data streams. Therefore, we propose a generic summary structure, respectively microcluster structure, that captures all the necessary statistical information of the incorporated data points during the online phase. In the offline phase, which can be initiated on demand or periodically, the information stored in the microclusters can be reused by any *PCA*-based clustering technique to generate a final correlation clustering model. As aging mechanism, a damped window model allows to "forget" the information that is contributed by stale data and, thus, keeps the microcluster model up-to-date.

Considering the state-of-the-art (see Sections 3.1 and 3.2), we briefly want to summarize that there exist a variety of approaches for both correlation clustering and stream clustering. While several correlation clustering algorithms on static data sets draw on *PCA*-based solutions but require multiple scans over the entire data set, stream clustering approaches mainly use appropriate data aggregation structures to be able to deal with the potentially infinite volume of data delivered by data streams. However, to the best of our knowledge, there currently does not exist a solution that solves the combination of both problems, which is a streaming framework able to construct correlation clustering models from streaming data where the data generating process reveals local and arbitrarily oriented subspace clusters within the data distribution.

In our experiments section, we investigate the runtime as well as the clustering quality of our approach and show that despite its improved performance compared to the static counterpart algorithms, the loss of accuracy is insignificant since the subspaces are mostly detected correctly. Furthermore, when considering the throughput of our approach we can state that the algorithm can cope with high-velocity data streams that push new data objects within milliseconds to the application.

4.2 A Generic Aggregation Structure for Correlation Clustering on Data Streams

For our purpose, the main issue is to find an appropriate data aggregation structure that captures sufficient statistics to represent the compressed information properly. At the same time it must be generic enough so that it can be used for any *PCA*-based correlation clustering technique. A major criterion for such a microcluster structure is the ability to be processed in an incremental manner since we cannot afford to store every data point and recompute the statistics of the microcluster from scratch each time a new data point arrives from the stream. As the focus of this work lies on *PCA*-based correlation clustering, we therefore borrow an *incremental principal component analysis* approach from [163]. Our empirical evaluation shows that this technique can sufficiently cope with high velocity data streams.

In general, there are two categories of *IPCA* algorithms. The first one are covariance-free techniques that cope without the computation or reconstruction of the covariance matrix. The algorithms from the second category approximately reconstruct the covariance matrix from the previously computed eigenvectors and eigenvalues. By adding a new observation, the dimension of the subspace is increased by one. But since least significant principal components are discarded, the dimension of the subspace is kept small which makes these approaches computational efficient. Although these methods suffer from unpredicted approximation errors, we use the basic algorithm presented in Algorithm 1 [163] because of its good real-time performance. Algorithm 1 proceeds as follows: First, after collecting an initialization set $\mathcal{I}$ of *init* observations, an initial *PCA* yielding the eigenvector matrix V_0 and eigenvalue matrix E_0, is performed on $\mathcal{I}$. The initial mean value is set to be the mean of all observations in $\mathcal{I}$. After the initialization, the initial eigenspace is updated incrementally with each incoming observation from data stream $\mathcal{S}$. Whenever a new observation x_i' arrives, it gets mean normal-

Algorithm 1 Incremental PCA

```
Input: Data Stream $\mathcal{S}$, Weight parameter $\tau$
Output: Current Eigensystem $eig_i$, composed of eigenvector matrix $V_i$ and
    eigenvalue matrix $E_i$
 1: $V_0, E_0 :=$ initial PCA from the first *init* observations
 2: $\mu_0 :=$ mean of the first *init* observations
 3: while $\mathcal{S}$ does not end do
 4:     $x'_i :=$ next incoming observation from $\mathcal{S}$
 5:     $x_i = x'_i - \mu_{i-1}$
 6:     $\mu_i = \mu + (1-\tau) \cdot x_i$
 7:     for $j \in range(0, col(V_{i-1}))$ do
 8:         $y_i = \sqrt{\tau E_{i-1}(j,j)} \cdot V_{i-1}(:,j)$
 9:     end for
10:     $y_{col(V_{i-1})} = \sqrt{1-\tau} \cdot x_i$
11:     $A = [y_0, y_1, ..., y_{col(V_{i-1})}]$
12:     $B = A^T A$
13:     $U, E_i =$ Eigen-decompose $B$
14:     for $j \in range(0, col(U))$ do
15:         $v_j = A \cdot U(:,j)$
16:     end for
17:     $V_i = [v_1, v_2, ..., v_{col(U)}]$
18: end while
```

ized, i.e. x_i, and the current mean μ_i is determined. The parameter $\alpha \in [0,1]$ is used as a weight that denotes the importance of a new observation compared to the previously seen ones. The larger α, the less important is a new observation x_i. Next, a $d \times col(V_{i-1}) + 1$ matrix A is defined, with $col(V_{i-1})$ denoting the number of columns of the eigenvector matrix V_{i-1}. The first $col(V_{i-1})$ columns of A are constructed from the weighted previous principal components and the weighted current observation forms the last column. Using matrix A, we can reconstruct the new $d \times d$ covariance matrix C expressed by $C = AA^T$. Since a high dimension d leads to high computational costs, a smaller $(col(V_{i-1}) + 1) \times (col(V_{i-1}) + 1)$ matrix $B = A^T A$ is constructed and then decomposed on the rank of $col(V_{i-1}) + 1$. The eigen-decomposition retrieves the eigenvalue matrix E_i and the eigenvector matrix U. Multiplying each eigenvector of U with matrix A finally retrieves the eigenvectors of the covariance matrix C with the eigenvalues contained in E_i. For the mathematical derivations of the single steps, we refer to [163].

The Correlation Clustering Microcluster Structure CCMicro

As this part of the thesis concentrates on streaming data, we use a common definiton of data streams.

Definition 1. *A data stream $\mathcal{S}$ is an ordered and possibly infinite sequence of data objects $x_1, x_2, ..., x_i, ...$ that must be accessed in the order they arrive and can be read only in one linear scan.*

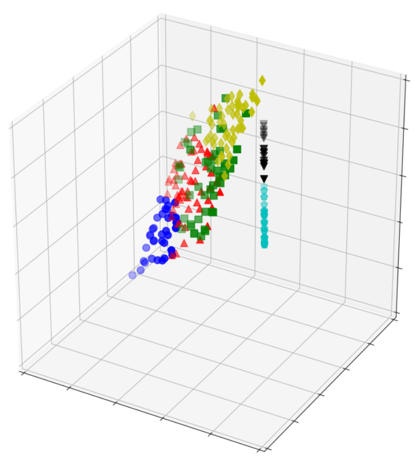
Another concept used for our purpose is the *damped window model.* Since recent data is typically more important than old data objects, especially if an up-to-date clustering model is desired, it is useful to "forget" stale data. Therefore, a widely used approach in applications dealing with temporal data is the utilization of the *exponential fading function* for data aging. This technique assigns a weight to each data object which decreases exponentially with time t by using the fading function $f(t) = 2^{-\lambda \cdot t}$. $\lambda > 0$ is the decay rate and determines the impact of stale data to the application. A high value of λ means low importance of old data and vice versa.

As we rely on the concept of microclusters, we need to define a data structure that encapsulates the necessary information and simultaneously allows update procedures. To fulfill these properties, we define our microcluster structure, which is called *CCMicro*, as follows:

Definition 2. *A microcluster CCMicro at time t for a set of d-dimensional points $\mathcal{C} = \{p_1, p_2, ..., p_n\}$ arriving at different points in time is defined as CCMicro$(\mathcal{C}, t) = (V(t), E(t), \mu(t), ts)$ with*

- *$V(t)$ being the eigenvector matrix of the covariance matrix of $\mathcal{C}$ at time t,*
- *$E(t)$ being the corresponding eigenvalues of the eigenvectors in $V(t)$,*
- *$\mu(t)$ being the mean of the data points contained in $\mathcal{C}$ at time t, and*
- *ts being the timestamp of the last incoming object assigned to this microcluster.*

Let us note that we generally differ between *strong eigenvectors* and *weak eigenvectors* in the eigenvector matrix V. The strength of an eigenvector is given by the variance along the corresponding axis in the eigenspace. We define *strong* and *weak eigenvectors* as follows [5]:


(a) Microcluster model.

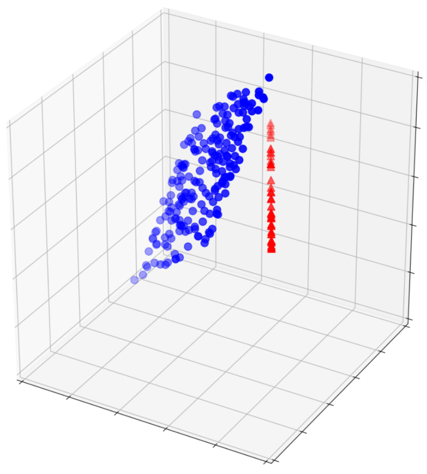

(b) Macrocluster model.

Figure 4.1: Micro- and macrocluster models on a toy dataset as retrieved by CorrStream. Clusters are depicted by plotting their point sets. Differently colored and shaped point sets describe different micro- resp. macroclusters.

Definition 3. *Given $\alpha \in [0, 1]$ and some microcluster mc. Let E_{mc} be the microcluster's $d \times d$ eigenvalue matrix having the eigenvalues in descending order on the diagonal. We call the first*

$$min_{r \in \{1,\dots,d\}}\{r | \frac{\sum_{i=1}^{r} E_{mc}(i,i)}{\sum_{i=1}^{d} E_{mc}(i,i)} \geq \alpha\}$$

eigenvectors strong eigenvectors resp. preference vectors and the remaining eigenvectors are called weak eigenvectors. The space spanned by the preference vectors is called correlation subspace.

Online Maintenance of CCMicro Structures

The generic CorrStream framework generally consists of two phases, i.e., an online phase in which microclusters are generated, maintained and/or discarded due to temporal expiration, and an offline phase to extract on demand clustering models of the current state of the stream. During the continuous online phase, which is outlined in Algorithm 2, the data stream is consumed and for each data object o a *rangeNN* query is performed to detect the closest microcluster. The *rangeNN* query retrieves the closest microcluster with a maximum distance of ϵ. If such a microcluster exists, it absorbs the current data object o, otherwise a new microcluster is created. Beside of the components that fulfill the maintenance properties, each microcluster has an initial buffer. This buffer is a small collection of data objects that serves as a basis for an internal initialization step. The intuition behind that is to collect a bunch of spatially close data objects for which an initial PCA is performed. The PCA retrieves the eigenspace of those data objects. Applying Definition 3, we can define the strong eigenvectors of the microcluster which span the correlation subspace.

Algorithm 2 Online

```
Input: Data Stream $\mathcal{S}$, range parameter $\epsilon$, buffer size $buff\_size$, decay parameter $\lambda$
1: for incoming data object $o$ from $\mathcal{S}$ at time $t$ do
2:    Microcluster $mc_{NN}$ = rangeNN$(o, \epsilon)$
3:    if $mc_{NN} \neq null$ then
4:       add $\langle o, t \rangle$ to $mc_{NN}$
5:    else
6:       create new microcluster with parameters $buff\_size$, $\lambda$ and add $\langle o, t \rangle$
7:    end if
8: end for
```

Note that the *rangeNN* query uses two distance measures, i.e., the Euclidean distance and the correlation distance. The reason for that is the period of grace that we establish for each newly constructed microcluster for the initialization. If the initial PCA has not been done for a microcluster yet, the correlation measure cannot be applied due to the lack of the microcluster's eigenspace. Therefore, we determine the Euclidean distance between the micocluster's mean point and the incoming data object instead of the correlation distance in such cases. However, to define the correlation distance, which is used in all other cases, we need to define the notion of *similarity matrix* beforehand.

Definition 4. *Let V_{mc} be an eigenvector matrix with E_{mc} being the corresponding eigenvalue matrix of a microcluster mc having onto* $[0; 1]$ *normalized eigenvalues on the diagonal. Given a threshold value $\alpha \in$* $[0; 1]$ *and a constant value $\kappa \in \mathbb{R}$ with $\kappa \gg 1$, the eigenvalue matrix E_{mc} is adopted by setting those eigenvalues to κ whose value is below the threshold value α. The values of the resulting matrix $\hat{E}_{mc}$ are computed according to the following rule:*

$$\hat{E}_{mc}(i,i) = \begin{cases} 1 & \textit{if } E_{mc}(i,i) \geq \alpha \\ \kappa & \textit{else.} \end{cases}$$

Having the adopted eigenvalue matrix $\hat{E}_{mc}$, the similarity matrix of mc is defined as

$$\hat{M}_{mc} = V_{mc}\hat{E}_{mc}V_{mc}^T.$$

The constant value κ specifies the allowed degree of deviation from the correlation subspace. Following [42], we set this value to $\kappa = 50$. The correlation distance can finally be computed as follows.

Definition 5. *Given a microcluster mc with mean point μ_{mc} and a data object o, the correlation distance between both is defined as*

$$distance_{corr}(mc, o) = \sqrt{(\mu_{mc} - o) \cdot \hat{M}_{mc} \cdot (\mu_{mc} - o)^T}$$

with $\hat{M}_{mc}$ being the similarity matrix of mc.

After determining the closest microcluster mc of the incoming data object o, the latter must be incorporated into mc properly. Our proposed algorithm basically differentiates three cases of how to insert a new data object into an existing microcluster. The first two cases are considered if the microcluster mc has not been initialized so far. In that cases, the object is inserted into the buffer and the mean as well as the current timestamp of the microcluster are updated. If the microcluster's buffer still has capacity, the insertion terminates by retrieving the updated microcluster. Otherwise, if the buffer is filled, the initial PCA is performed on the data objects contained in the buffer and the eigensystem is retrieved. After setting the corresponding components of the microcluster structure, mc is marked as initialized. The third option of inserting a new data object is used if the microcluster already has been initialized. In this case, the existing components of the microcluster are reused and the incremental PCA procedure is invoked to generate the new eigenvectors and -values as well as an updated mean vector. As mentioned above, the degree of influence of the new object on the existing eigensystem can be regularized by the weight parameter.

Due to the possibility of expiring microclusters, i.e., microclusters that have not absorbed any data object for a while, it might happen that this microcluster should be deleted since stale data should not sophisticate an up-to-date clustering model. Deleting old microclusters also has the advantage to safe storage space. As a straightforward solution, we propose to scan the set of microclusters sequentially from time to time and delete those microclusters whose timestamp ts_{mc} is older than a user specified threshold value Δt, i.e., if $ts_{mc} < ts_{curr} - \Delta t$ with ts_{curr} denoting the current point in time. Note that the choice of an appropriate threshold value Δt depends on the application at hand.

4.3 Application of CCMicro Structures for Offline Correlation Clustering

The main goal of the offline phase of CorrStream is the construction of a high quality clustering model that describes correlations appearing in the data. For that purpose, the algorithm chosen for the offline routine must be capable of building macroclusters on top of the generated *CCMicro* structures retrieved by the online process. Figure 4.1(a) exemplary depicts the outcoming microcluster model of the online phase for a small synthetic and 3-dimensional data set.

As can be seen easily, some of the microclusters can be grouped so that finally two separated macroclusters, i.e., an 1-dimensional cluster and a 2-dimensional one, are formed. In general, the microcluster structure is generic enough so that a variety of static correlation clustering algorithms can be adopted to build a clustering model based on the retrieved microclusters.

Integration into ERiC

In this section, we discuss a variant of the *ERiC* algorithm [5] in detail. The algorithm consists of four steps, i.e., (1) partitioning the set of microclusters, (2) computing macroclusters within each partition, (3) building the hierarchy of the clustering and (4) defining cross-partition macroclusters which is optional. In the first step, we partition the set of microclusters according to the dimensionality of their subspaces. Regarding the example data set from Figure 4.1(a), the microclusters which form the 1-dimensional line cluster would be in one partition, and the microclusters which form the 2-dimensional plane would be in another partition for instance. Technically, the partitioning process is done by counting the number of strong eigenvectors contained in the eigenvector matrix of each microcluster. After dividing the set of microclusters into disjoint partitions, the algorithm determines macroclusters within each partition. Therefore, we apply a *DBSCAN* [88] variant capable of dealing with the structure of the microclusters. The basic idea is to use the orientation of the microclusters given by the eigenvectors to group those microclusters whose eigenvectors span a similar subspace. According to [5], we define the correlation distance used for the spatial queries in the offline phase as the composition of the *approximate linear dependency* and the *affine distance* between two microclusters.

Definition 6. *Given a threshold value $\Delta \in \,]0;1[$ and two microclusters mc_i and mc_j, with mc_i having less or as many strong eigenvectors as mc_j, mc_i is* *called approximately linear dependent from mc_j if*

$$\sqrt{v_i^T \cdot V_{mc_j} \cdot \hat{E}_{mc_j} \cdot V_{mc_j}^T \cdot v_i} \leq \Delta \ ,$$

with $\hat{E}_{mc_j}$ being the adopted eigenvalue matrix of mc_j according to

$$\hat{E}_{mc}(x,x) = \begin{cases} 1 & \text{if } E_{mc}(x,x) \geq \alpha \\ 0 & \text{else,} \end{cases}$$

holds for all strong eigenvectors v_i of mc_i.

Note that the threshold value Δ is introduced to allow a certain degree of deviation from an absolute linear dependency between two microclusters.

Definition 7. *Let mc_i and mc_j be two microclusters with mc_i having less or as many strong eigenvectors as mc_j and let mc_i be approximately linear dependent from mc_j. Then, the affine distance between mc_i and mc_j is defined as*

$$dist_{aff}(mc_i, mc_j) = \sqrt{(\mu_{mc_i} - \mu_{mc_j})^T \cdot V_{mc_j} \cdot \hat{E}_{mc_j} \cdot V_{mc_j}^T \cdot (\mu_{mc_i} - \mu_{mc_j})},$$

with $\hat{E}_{mc_j}$ being defined as in Definition 6 and μ_{mc_i}, resp. μ_{mc_j}, being the mean of microcluster mc_i, resp. mc_j.

Combining Definitions 6 and 7, and assuming the premise that two microclusters with parallel subspaces form a joint cluster if the affine distance is below a threshold value $\delta \in \mathbb{R}_0^+$, we define the correlation distance for the offline phase as follows.

Definition 8. *Let $\delta \in \mathbb{R}_0^+$, $\Delta \in (0;1)$ and let mc_i and mc_j be two microclusters with mc_i having less or as many strong eigenvectors as mc_j. The correlation distance between mc_i and mc_j, i.e., $CorrDist_\Delta^\delta(mc_i, mc_j)$, is defined as*

$$CorrDist_\Delta^\delta(mc_i, mc_j) = \begin{cases} 0 & \text{if } mc_i \text{ is approx. linear dependent from} \\ & mc_j \wedge dist_{aff}(mc_i, mc_j) \leq \delta \\ 1 & \text{else.} \end{cases}$$

Using this distance measure for *DBSCAN* which is performed within each partition finally yields a density-based correlation clustering model for each partition. Note that the ϵ parameter for *DBSCAN* must be set to 0 since the distance measure is binary. The $minPts$ parameter, that we refer to as $minMcs$ in the following to be able to distinguish this parameter from the one used in the static *ERiC* method, depends on the application.

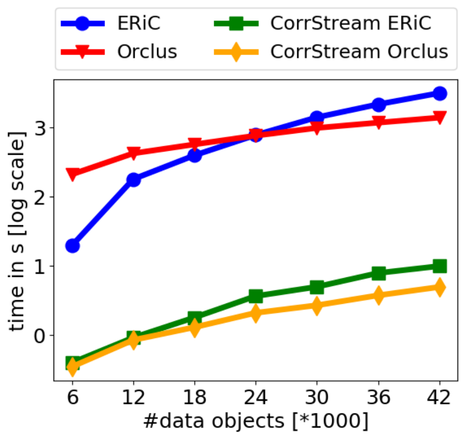
Due to the partitioning, it might happen that some microclusters might span a subspace that lies within the subspace of surrounding microclusters whose subspaces have a higher dimensionality, e.g., a line cluster might be embedded into a plane shaped cluster. To detect such constellations, step (3) builds a hierarchy of the macroclusters generated during the previous step. The procedure borrowed from [5] iterates over the set of macroclusters and checks for each cluster $\mathcal{C}_m$ whether there exists another cluster $\mathcal{C}_n$ whose subspace is of a higher dimensionality and for which $CorrDist^{\delta}_{\Delta}(\mathcal{C}_m, \mathcal{C}_n) = 0$ holds. If such cluster $\mathcal{C}_n$ exists, it is regarded as a parent of $\mathcal{C}_m$ unless $\mathcal{C}_n$ is not an ancestor of $\mathcal{C}_m$ already. Processing all macroclusters in this manner yields to a hierarchical, tree-like structure in which clusters contained in a child node are embedded within the clusters contained its parent node. In the optional fourth step of our algorithm, we finally merge related macroclusters which leads to cross-partition macroclusters, i.e., clusters that contain microclusters which span subspaces of different dimensionalities with the premise that lower dimensional microclusters must be embedded within the microclusters of higher dimensionalities.

Figure 4.1(b) shows the final clustering result of our sample data set. Note that full dimensional subspaces are treated as noise, depicted as blue dots. Lower dimensional microclusters that obviously lie within a cluster can turn out to be handled as noise as well if they are not *density-reachable* for the *DBSCAN* algorithm. Regarding the lower right corner of the 2-dimensional cluster gives an example for this phenomenon. Although there exists a microcluster in the model retrieved by the online phase, this microcluster is labeled as noise during the offline phase since the microcluster is 1-dimensional but not density-reachable from another 1-dimensional microcluster (and vice versa) to satisfy the $minMcs$ criterion for *DBSCAN*.

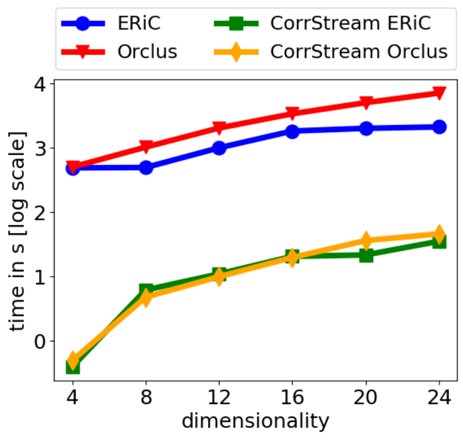
Integration into ORCLUS

As for *ERiC*, the adaptations that have to be made for the *ORCLUS* algorithm are quite small. First, the parameter $k0$, i.e., the initial number of seeds that the algorithm starts with, has to be set to a fraction of the number of microclusters since the user usually do not know the exact number in advance. Further, as the algorithm basically has to work with the mean points of the microclusters as well as their eigenvectors, we propose to incorporate the orientations of both affected eigensystems when measuring the distance between two microclusters. If just assigning by measuring the projected pairwise distance of the mean points, as given in [17], it might happen that two microclusters that have a different subspace orientation are grouped because the mean of the one microcluster fits into the subspace of the other one although their subspace preferences are different.

4.4 Experiments



(a) Runtime experiments for various database sizes.



(b) Runtime experiments for varying dimensionalities.

Figure 4.2: Runtime experiments for varying DB sizes and dimensionalities. Note the log scale on the y-axes.

For our experimental evaluation, we simulate vector data sets as data streams by processing the data objects one after the other.

We compare our proposed CorrStream algorithm with the static equivalents, i.e., the *ERiC* and the *ORCLUS* algorithms, since we use a variant of both methods for the offline phase. Therefore, we generate labeled data and compare the precision and recall values of both methods. We compare the considered approaches in terms of runtimes as well as in terms of clustering quality. We consider different database sizes, respectively various numbers of objects delivered by the data stream, as well as various dimensionalities of the data points. Unless differently stated, all data points are distributed among 5 equi-sized clusters. For the experiments using variously sized data sets, each cluster has a random dimensionality between 1 and 2, and one full-dimensional noise cluster spanned over the entire normalized $\mathbb{R}^3$ space. For the experiments that investigate the performance under varying dimensionalities, the number of data points is fixed to 12'000 and except of one full-dimensional noise cluster, the correlation clusters have a random dimensionality which is below the full dimensionality.

For all experiments, we report the results when using the best considered parameter setting. We consider different parameter settings by performing a grid search over buff_size $\in \{10, 15, 20\}$; $\epsilon \in \{0.1, 0.15, 0.2, 0.3\}$; $minMcs \in \{1, 2, 3\}$. The parameters that were already introduced by previous methods are fixed.

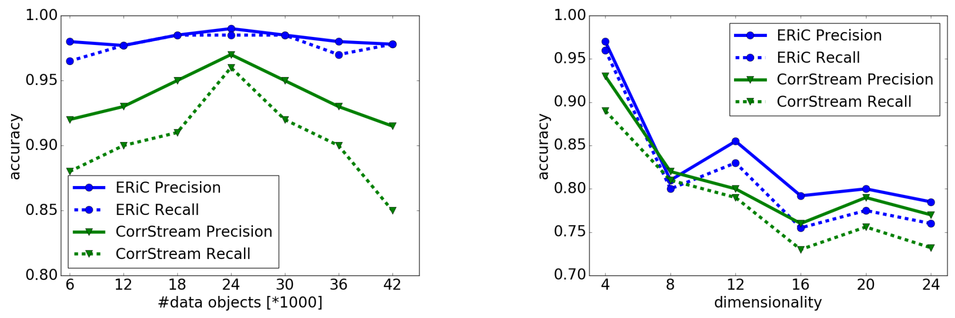
Runtime Experiments

The first set of experiments investigate the performance of the proposed method in terms of runtime. Figure 4.2(a) shows the results for varying numbers of data objects. The y-axis shows the measured runtime in log scale. While the static methods both show rather fast increasing runtimes, even for moderately sized databases, the CorrStream variants are able to process 42'000 data objects within only a few seconds. When ranging the dimensionality of the feature space from 4 to 24 dimensions, the outcome is quite similar. As depicted in Figure 4.2(b), the static algorithms need much more time compared to our method. Again, as can be observed, the CorrStream variants significantly improve the static competitors and only need a few seconds.

Quality Experiments

The next set of experiments investigates the considered methods in terms of clustering quality. Therefore, we measure the quality by comparing the resulting clusterings of each approach to the ground truth labels. Precisely, we use precision and recall values to examine the performance. First, Figure 4.3 compares the results of CorrStream using the *ERiC* approach for the offline phase against the static *ERiC* algorithm. In both plots the *ERiC* algorithm shows higher precision and recall values than CorrStream. This might be reasoned by aging as well as by treating microclusters as noise if their buffer have not been filled. Figure 4.4 shows the corresponding results when using *ORCLUS* for the offline procedure. Interestingly, the CorrStream results are better in the experiments when increasing the dimensionality. An explanation for this result might be the phenomenon that during the first iteration of *ORCLUS*, all data points are assigned to the closest cluster center with respect to the Euclidean distance. This leads to a situation where clusters, resp. groups of points that are treated as intermediate clusters during the iterations, are spanned across several actual clusters.

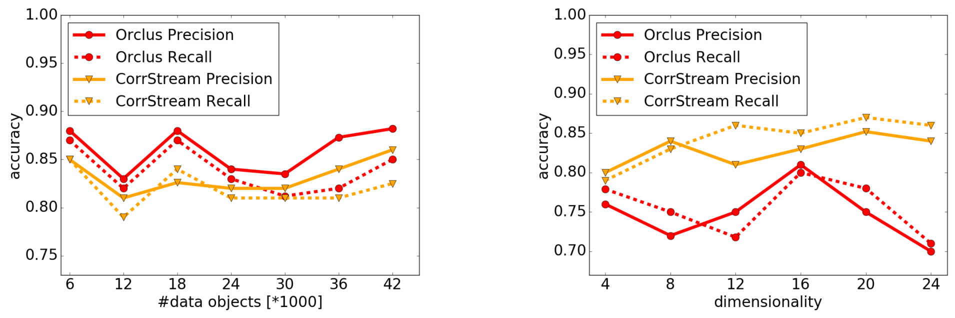
Such intermediate clusters, that typically occur if differently oriented correlation clusters intersect, finally form a “false" subspace and thus might absorb data points that actually do belong to another cluster. CorrStream reduces this problem due to using the correlation distance for the assignment of a data point as soon as the initialization of a microcluster is completed.



(a) Results for varying numbers of data objects.

(b) Results for varying dimensionalities.

Figure 4.3: Precision and Recall measurements when considering *ERiC*.



(a) Results for varying numbers of data objects.

(b) Results for varying dimensionalities.

Figure 4.4: Precision and Recall measurements when considering *ORCLUS*.

Throughput

Finally, as the throughput is one of the major criterions for streaming algorithms, we evaluate CorrStream in terms of throughput, i.e., the number

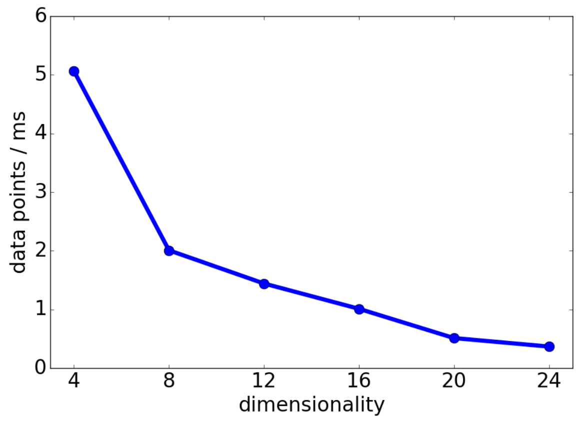


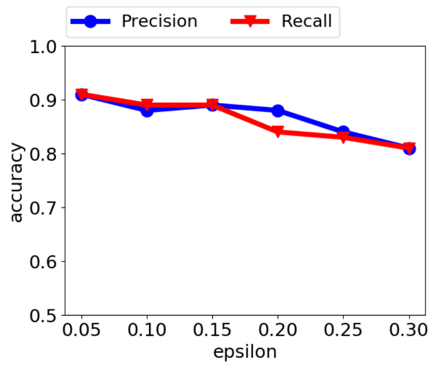
Figure 4.5: The throughput of the online phase by considering different dimensionalities.

of data objects processed per millisecond, by using various dimensionalities. The plot in Figure 4.5 investigates the throughput of the online phase by using different dimensionalities for the feature space. As can be seen, the throughput of the algorithm decreases with increasing dimensionalities. Nevertheless, the decline of the throughput decreases for higher dimensionalities and still is about 363 data objects per second for 24 dimensional feature spaces.

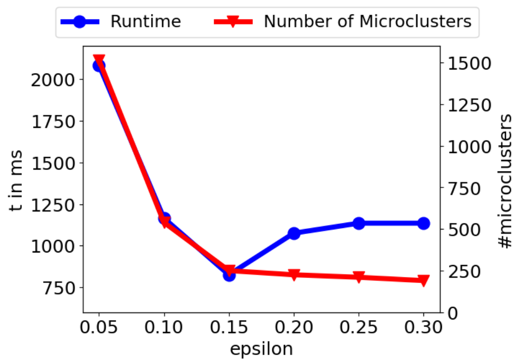
Influence of Parameters

In the following block of experiments, we discuss the influence of the parameters that are newly introduced in this work and are required to be set by the user for CorrStream, i.e., the ϵ parameter giving the allowed maximum distance of a data point to a microcluster center for the assignment step, and the *buff_size* parameter which regulates the size of the buffer, or in other words the number of points used for the initial PCA. Note that we omit the discussion of the *minMcs* parameter, as this parameter is *ERiC* specific and is not used in the *ORCLUS*-like variant. Note that the *ORCLUS* specific k_0 parameter, i.e., the parameter that defines the initial number of clusters which is subsequently reduced to k during the iterations, is implicitly given by the number of microclusters.

The plot in Figure 4.6 describes the influence of the ϵ parameter. The precision and recall values slightly decrease with increasing ϵ values, cf., Figure 4.6(a). This can be explained by the enlarged absorption radius as more distant points can be absorbed by a microcluster, especially during the initialization phase. Furthermore, as can be seen in Figure 4.6(b), the number of produced microcluster decreases with increasing values of ϵ and thus the required computation time, too.

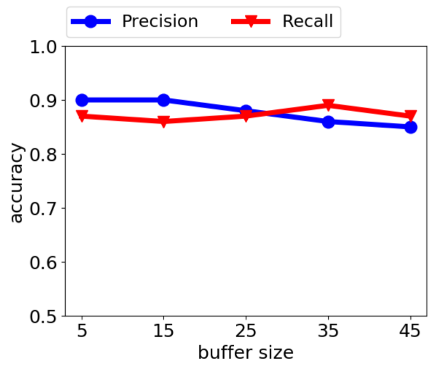


(a) Precision and Recall for varying values of ϵ.

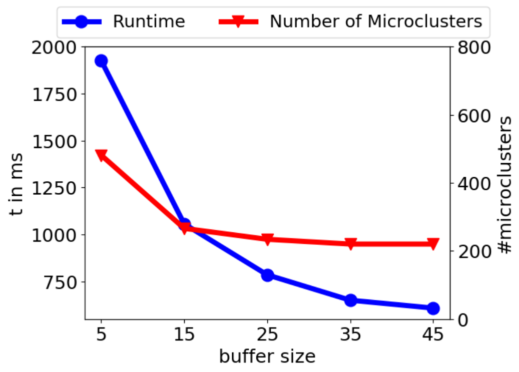


(b) Runtime (left axis) and the absolute number of microclusters (right axis) for varying values of ϵ.

Figure 4.6: Performance measures for various values of the ϵ parameter.



(a) Precision and Recall for varying values of *buff_ size*.



(b) Runtime (left axis) and the absolute number of microclusters (right axis) for varying values of *buff_ size*.

Figure 4.7: Performance measures for various values of the *buff_ size* parameter.

Figure 4.7 shows the results when considering various buffer sizes. While varying values for this parameter hardly affect the measured precision and recall values, the measurements for the runtime shows that the size of the initial microcluster buffers has a rather high impact on the performance, which generally seems to benefit from larger buffers.

To summarize: based on our findings we want to state that, of course, the choice of the parameters affects the efficiency of the online phase. An inappropriate selection of parameters might lead to increased runtimes. For instance if the parameter ϵ is chosen wrongly, the number of generated microclusters may increase which in turn leads to a higher number of necessary distance computation when assigning incoming data objects to existing microclusters. Note that the increased computational costs do not necessarily lead to a significany improvement for the overall clustering quality. On the other hand, it might happen that the combination of the ϵ and *buff_size* parameters are chosen in such a way that microclusters do not initialize. However, this strongly depends on the data distribution that is given by the underlying data generating process.

4.5 Conclusion

We presented a novel streaming algorithm capable of detecting arbitrarily oriented subspace clusters, i.e., correlation clusters, in this chapter. Regarding the methodology, we applied the established two-step approach by dividing the procedure in an online and an offline phase. A newly proposed microcluster structure is used to aggregate similar data objects and thus compressing the volume of data significantly. At the same time this data structure provides all the necessary statistical information gained from the incorporated data points that are required for the offline phase to compute a clustering model which represents the structure of the data in total sufficiently. Our experimental evaluation showed that our streaming algorithm outperforms its static counterparts clearly in terms of computational costs by just suffering a small loss concerning the clustering quality. Furthermore, we could measure a satisfying throughput of up to several hundred data objects within seconds.

However, since our method needs several user-specified parameters for which a fundamental grasp of the internal structure of the data is advantageous, a parameter estimation mechanism would be an interesting topic for future work. Another point worth to investigate would be an extension to enable the detection of non-linear correlation clusters. As the microcluster structure allows to extract the information about the orientation of the subspaces we see the potential that this can be used to detect non-linear correlation within the data as well.

Chapter 5

Detecting Linear Correlated Clusters on Streams using Parameter Space Transform

The work presented in this chapter is going to appear in the Proceedings of the 24th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2020.

5.1 Introduction

Data clustering is an established and widely used technique for explorative data analysis and in general a decent approach for tackling many unsupervised problems. However, when facing high-dimensional data, particularly clustering algorithms quickly reach the boundaries of their usefulness as most of these methods are not designed to deal with the curse of dimensionality. Due to inherent sparsity in high-dimensional data, distances between objects tend to become meaningless since the distances between any two objects measured in the full dimensional space tend to become the same for all pairs of objects. This is a serious problem for most clustering algorithms since they mostly rely on distance calculations to distinguish similar from dissimilar objects. Furthermore, clusters often appear within lower dimensional subspaces with the subspaces including various dimensions and dimensionalities. Therefore, it may not be useful to search for clusters in the full dimensional data space since even if clusters are detected they hardly retrieve any insights for the user due to lack of interpretability. To overcome those issues, several subspace clustering algorithms have been developed in the past. All subspace clustering algorithms generally have the objective to (1) identify meaningful subspaces and (2) detect clusters within these subspaces. They can be categorized into two groups: projected clustering and oriented subspace clustering algorithms (cf. Section 3.1). Projected clustering algorithms restrict themselves to the detection of axis-parallel subspace clusters. Oriented subspace clustering algorithms allow the combination of features (i.e., the original dimensions of the data space) to identify new (and interesting) dimensions which may form a lower dimensional subspace in which clusters can be identified easily by applying conventional clustering approaches[1]. While clusters found in projected subspaces are generally easier to interpret for the end-user, oriented subspace clustering provides a better clustering in many applications since the assumption that features are independent from each other (as inherently assumed by projected clustering approaches) usually does not hold.

In this chapter, we present a novel oriented subspace clustering algorithm that is able to detect arbitrarily oriented subspace clusters in data streams. As discussed in the previous chapter, data streams implicate the challenge that the data cannot be stored entirely and hence there is a general demand for suitable data handling strategies for clustering algorithms such that the data can be processed within a single scan. In contrast to the CorrStream algorithm from Chapter 4, the method presented here relies on the Hough transform and finds global, arbitrarily oriented subspace clusters rather than local correlation clusters derived from neighborhood sets. Renouncing the usage of neighborhood sets has the big advantage of being less dependent from outliers, resp. noise data, as they might appear in vast numbers within neighborhoods sets, especially when considering high-dimensional data. In general, the method presented here can be understood as a streaming variant of the *CASH* algorithm [3] which has been designed for robust correlation clustering in static data. However, when looking for relevant subspaces, *CASH* performs a top-down, grid-based data space division strategy with the idea to prune sparse grid cells. This is inappropriate when considering a data stream where the data distribution may change over time. On the other hand, dense grid cells may become sparse and thus irrelevant over time, too. Therefore, we propose a batched variant that is able to deal with those challenges. The key idea is to load chunks of data into memory, deriving so-called *concepts* as summary structures and applying a decay mechanism to downgrade the relevance of stale data. Our experimental evaluation demonstrates the usefulness of the presented method and shows that the used heap space is drastically reduced without losses in terms of runtime and accuracy. It is noteworthy that the complexity discussion even states that the runtime

[1]These are typically partitioning or density-based clustering techniques.

can be reduced for sufficiently large datasets.

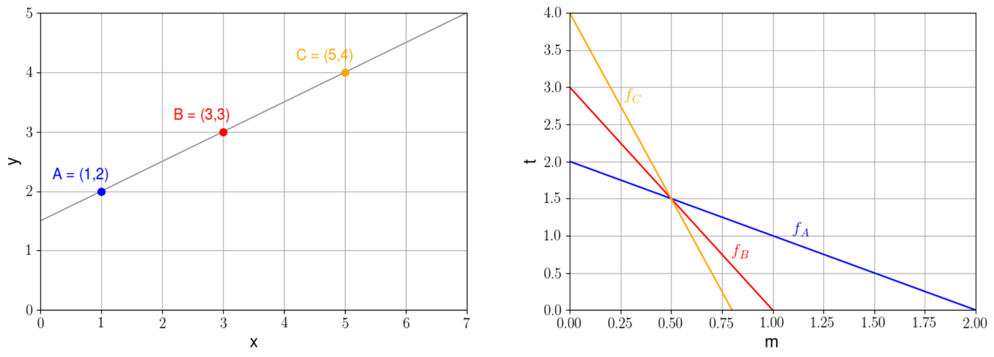
The remainder of the chapter is as follows. In Section 5.2 we recapitulate the basic principles behind correlation clustering using Hough transform since our proposed method heavily relies on the techniques used there for identifying correlation clusters. Then we propose our algorithm that is able to follow this paradigm in a streaming environment in Section 5.3. The experimental evaluation is shown in Section 5.4, and Section 5.5 finally gives a discussion on our findings and concludes the chapter.

5.2 Correlation Clustering Using Parameter Space Transformation

Hough Transform

The Hough Transformation originally has been introduced as a useful tool for image processing in [218]. The basic idea is to translate every pixel from image space into a straight line in parameter space, and every intersection of two lines in parameter space means that these two points are on a straight line in image space. This way, one can determine straight lines and, more broadly, linear segments in the data space simply by detecting areas where many lines intersect in parameter space, which is an important task in image processing. More formally, for a data point $p = (x, y)$ in a two-dimensional data space $\mathcal{D}$, one can define a parameter space, once again two dimensional, $\mathcal{P}$ with axes m and t such that that p is represented by the straight line $t = -xm + y$, with x being the negative slope and y denoting the axis intercept. This way, if there is a point $S = (s_x, s_y) \in \mathcal{P}$ where several parameter space lines l_i intersect, there is a common line $y = s_x x + s_y$ in $\mathcal{D}$ that goes through all of the inverse image points of the lines l_i. The idea is visualized in Figure 5.1. Given the three data points A, B and C in data space (left), we can transform them into linear functions in parameter space (right). Since A, B and C are perfectly correlated, the lines in parameter space intersect at one point in parameter space. Reconstructing this point in the data space retrieves a linear function on which all data points A, B and C lie (grey line).

However, using straight lines in parameter space has a significant drawback, namely that the slopes of the lines are unbounded, i.e., they may be infinite, and therefore the possible intersection points of lines in the parameter space can not be controlled. A solution to this has been introduced in [83], where polar coordinates are being used in the parameter space. In fact, using angles and radii for parameters, and trigonometrical functions in-



(a) Cartesian space (b) Parameter space

Figure 5.1: Left: data space, right: parameter space

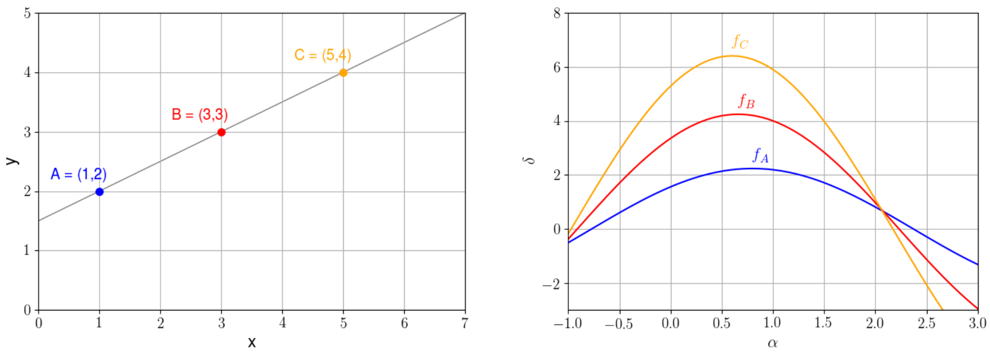
stead of straight lines effectively avoids the problem stated above. For a data point $p = (x, y)$ in a two-dimensional data space the point is mapped into parameter space by the trigonometrical function $\delta = x\cos\alpha + y\sin\alpha$, with axes α and δ. Note that the free parameter α is bounded within the interval $[0, \pi)$. These functions will be called *sinusoid* in the following. Analogously to the originally proposed parameter space, it holds that if such sinusoids intersect in a point $S = (\alpha_s, \delta_s)$ in parameter space P, the inverse image points lie on a straight line l_S in the image space. Again, the corresponding line in image space can easily be derived from S since δ_s is the distance of the line to the origin, α_s corresponds to the angle between the x axis of $\mathcal{D}$ and the perpendicular line of l_S that intersects the origin. Similar to above, Figure 5.2 depicts the procedure when using the parameter space spanned by the parameters of the polar coordinate representation. Considering the right image, the functions are sinusoids rather than linear functions this time.

Using Hough for CASH

The sinusoidal parameterization function can be extended for the case where we need to deal with higher dimensional data. Given a d-dimensional data space $\mathcal{D} \subseteq \mathbb{R}^d$ and a point $p = (p_1, ..., p_d)^T \in \mathcal{D}$, the parameterization function $f_p : [0, \pi)^{d-1} \rightarrow \mathbb{R}$ is defined as

$$f_p(\alpha_1, ..., \alpha_{d-1}) = \sum_{i=1}^{d} p_i \cdot \left(\prod_{j=1}^{i-1} \sin(\alpha_j) \right) \cdot \cos(\alpha_i),$$

with $\alpha_d = 0$. This corresponds directly to the generalized polar coordinates of a vector [3]. The basis of the resulting parameter space is defined by



(a) Cartesian space (b) Parameter space

Figure 5.2: Left: data space, right: parameter space

the $d-1$ angles $\alpha_1, ..., \alpha_{d-1}$ and $\delta = f_p(\alpha_1, ..., \alpha_{d-1})$. Precisely, this means that any point p from data space can be mapped to some function in the d-dimensional parameter space with each point finally being represented by the angles of the normal vectors defining the hyperplanes in Hessian normal form, i.e., $\alpha_1, ..., \alpha_{d-1}$, and their distances to the origin, i.e., δ. In particular, this also means that a point $S = (\alpha_1, ..., \alpha_{d-1}, \delta)$ in parameter space stands for a $d-1$ dimensional hyperplane in the data space, while a function in parameter space corresponds to all possible $d-1$ dimensional hyperplanes that contain the data point p. The following important conclusion can be made: if the parametrization functions of two data points intersect in parameter space, the intersection point represents a hyperplane in data space containing both points. The same is true for any amount of points.

Taking this into account, intuition already suggests the idea of *CASH*, i.e, search for dense areas in the parameter space in order to find data points with common hyperplanes. A dense area is an area where many parametrization functions intersect each other, or to relax this a little, small partitions of parameter space which are intersected by many parameterization functions[2].

Recalling that *CASH* employs a top-down grid-based space partitioning strategy, the density criterion also introduces the first input parameter of the algorithm: *minPoints* or m. It specifies how many intersections are required doe a partition of the parameter space to be considered dense. The other user-specified input parameter is the size of the partitions of the parameter space, i.e., *maxSplits* or s. Since it is nearly impossible to calculate all

[2]Note that intersection points correspond to perfectly correlated subspace clusters. However, by relaxing this restriction to small areas in the parameter space, we allow the algorithm to identify correlation clusters that are not perfectly correlated, too.

possible intersection points of all parametrization functions, it is necessary to discretize the parameter space by a grid. The resulting grid cells, also called *cuboids*, act as the small partitions of parameter space. Obviously, it is necessary to know the range of all axes for such a grid-based strategy. For the α angles, the bounds are 0 and π. For the δ-axis, the minimum and maximum is equal to the minimum and the maximum over all extrema of all parametrization functions. Since every parametrization function f_p is a sinusoid with period of 2π, there is a global extremum $\tilde{\alpha} = (\tilde{\alpha}_1, ..., \tilde{\alpha}_{d-1})$ in $[0, \pi)^{d-1}$, which can be calculated by using the Hessian matrix of f_p. Depending on whether this is a maximum or a minimum, the opposite extreme value of the sinusoid on the domain can be calculated (see [3] for details). Finally, given the global minimum and maximum for every f_p, the boundaries $[d_{min}, d_{max})$ of the δ-axis are defined by

$$d_{\max} = \max_{p \in \mathcal{D}} \left(\max_{\tilde{\alpha} \in [0,\pi)^{d-1}} f_p(\tilde{\alpha}) \right),$$

$$\text{and } d_{\min} = \min_{p \in \mathcal{D}} \left(\min_{\tilde{\alpha} \in [0,\pi)^{d-1}} f_p(\tilde{\alpha}) \right),$$

and we can consider the domain of parameter space as $[0, \pi)^{d-1} \times [d_{min}, d_{max}]$.

If this domain is discretized into a grid of cuboids, the next step is to determine whether a sinusoid f_p intersects a cuboid C. This can be done by calculating the minimum $f_{p_{\min}}(C)$ and maximum $f_{p_{\max}}(C)$ of the sinusoid within the boundaries of the cuboid, and checking if the resulting interval overlaps the δ-interval of the cuboid $[d_{C_{min}}, d_{C_{max}}]$. If both conditions are met, the corresponding parametrization function intersects the cuboid C. However, this procedure may become computationally expensive, especially in higher dimensional data where grid-based approaches generally tend to be inefficient in terms of computational cost. In fact, searching for dense grid cells in a d-dimensional data space exhaustively would require $(2^n)^d$, with n being the amount of splits per dimension, grid cells to be examined. Therefore, *CASH* uses a division strategy that prunes grid cells to decrease the search space and hence reducing the complexity.

Division strategy

Starting with the full dimensional cuboid $C \in [0, \pi)^{d-1} \times [d_{C_{min}}, d_{C_{max}}] \subseteq \mathbb{R}^d$, and the pre-defined divison order $\delta, \alpha_1, ..., \alpha_{d-1}$, the algorithm first splits the cuboid into two halves along the δ-dimension. For the resulting two cuboids, the number of intersecting sinusoids is calculated. If the intersection count of one of the cuboids is higher than the input parameter m, the cuboid is divided recursively by the division order. If the count of intersections is above m for both cuboids, the second cuboid is pushed into a queue. If the number of intersections is less or equal than m, the cuboid can be discarded. Once a cuboid reaches the maximum split threshold s, the data objects of the corresponding sinusoids are considered to form a subspace cluster and the sinusoids are no further considered for the recursive search within the other cuboids. If the division process of a cuboid terminates (either because the maximum split threshold is reached or the cuboid is discarded due to sparsity) the next cuboid is taken from the queue and split recursively.

Finding lower dimensional clusters and hierarchies of clusters

Having found a cluster, respectively a dense cuboid $C \subseteq \mathbb{R}^d$ after s divisions, means that the corresponding points form a cluster within a $(d-1)$-dimensional subspace. However, the cluster might either be lower-dimensional or another lower-dimensional cluster can be embedded within the found subspace cluster. Therefore, the sinusoids that form the $(d-1)$-dimensional cluster are transformed back into the data space and projected onto the orthonormal basis that can be derived from cuboid C. Somewhat more precise, given the boundary intervals of C, the normal vector of the corresponding hyperplane in polar coordinates is defined by the means of the cuboid's boundary intervals. This normal vector is transformed back into the Cartesian data space and finally a $(d-1)$-dimensional orthonormal basis, which defines the subspace hyperplane[3], is derived from it. To detect subspace clusters of even lower dimensions, the *CASH* algorithm is performed on the resulting $(d-1)$-dimensional data set recursively until no more cluster can be found. It is worth to note that this procedure creates an implicit dimensional cluster hierarchy.

5.3 CashStream

Data Processing: Batch Processing

Intuitively, the process of adapting the subspace clustering procedure of *CASH* in a stream setting might seem straightforward: For every new data

[3]Note that the resulting subspace may have a small error since its orientation is defined by the cuboid's mean vector rather than the mean vector of the corresponding parts of the intersecting parametrization functions.

point, determine which cuboids are being intersected by its parameterization function. However, regarding the facts that data cannot be kept in memory entirely and stale data shall be downgraded within stream applications, this solution is inappropriate. The strongest argument against that approach is the way the division strategy works. At every depth of the division of cuboids, cuboids are being discarded and particularly, not being kept in memory. This means that only clusters that have already proven to correspond to dense cuboids at the lowest split level can potentially absorb an incoming data object. In particular, this also means that the creation of new clusters is impossible and in addition, potential concept shifts, i.e., abrupt changes in the underlying data distribution, will almost not be noticeable. One could argue for keeping certain cuboids in memory which are close to surpassing the *minPoints* threshold, but this closeness would introduce another parameter and the number of cuboids kept in memory could explode under certain conditions, for example for highly noisy data sets. Another way of tackling this would be to partition the parameter space into a static coarse grid and to reevaluate its dense cells with the division strategy for every new data point if the point changes the count of intersections of the cell. A major drawback of this solution is that it does not generalize well to data sets of different dimensionalities unless the granularity of the coarse static grid is another parameter to be defined. Also, for high-dimensional data, a seemingly coarse grid still has a high number of cells since it is exponential in regard to the dimension.

Regarding the requirements that a streaming approach should be able to deal with both changes in the underlying data distribution and data of different dimensionalities, we propose to process incoming data in batches. The idea is similar to the data processing scheme proposed in [104], i.e., loading chunks of data into memory and eventually computing cluster representatives which are kept in memory while the actual data objects are discarded to empty the space for upcoming data. On the one hand, this data processing scheme enables the recognition of concept shifts since processing an entire batch of data increases the probability of identifying dense grid cells (potentially with novel subspaces) during the division steps. On the other hand, it is also suitable in terms of processing data of different dimensionalities as batch processing allows for using the originally proposed heuristic for pruning sparse grid cells and hence there is no need for defining a static grid. Precisely, our algorithm basically performs a slightly adapted variant of *CASH* on a data chunk and keeps cluster representatives, which we will refer to as *Concepts* in the following, in memory. Since the *Concepts* must be maintained efficiently, they are designed to be additive, such that two similar *Concepts* can conveniently be unified into a single *Concept*. Algorithm 3 outlines the main procedure of CashStream.

Algorithm 3 CashStream

```
Input: Data Stream $\mathcal{S}$, Batch size $b$
 1: $Clustering = \emptyset$
 2: $batch$ = empty collection of size $b$
 3: for incoming data object $o$ from $\mathcal{S}$ do
 4:     if $batch$ is not full then
 5:         add $o$ to $batch$
 6:     end if
 7:     if $batch$ is full then
 8:         currentConcepts = $CASH(batch)$
 9:         $Clustering$.add(currentConcepts)
10:         $unifyConcepts(Clustering, ...)$          // see Algorithm 4 $batch$ = empty
            collection of size $b$
11:     end if
12: end for
```

Note that the batch processing scheme obviously requires the user to specify the size b of a batch which is a hyperparameter. However, our experimental evaluation shows that the size of a batch, if chosen realistically, is not strongly influencing the performance of the clustering process in terms of accuracy or runtime, but indeed caps the cost in terms of memory usage.

Cluster Representatives: *Concepts*

As a suitable summary structure for data objects that are assigned to a cluster we define a *Concept* as follows.

Definition 9. *A Concept is a data structure that defines a minimalistic clustering model, an abstraction of a cluster resulting from CASH. In a d-dimensional data space $\mathcal{D}$, a Concept of dimensionality $l < d$ captures a l-dimensional hyperplane in parameter space $\mathcal{P}$ with rudimentary information of the data objects it contained as a result of CASH. A Concept consists of the following attributes:*

- *a set E containing $d - l$ equations in Hesse normal form,*
- *mean μ of all data objects that are assigned to the cluster,*
- *number of data objects N that are assigned to the cluster,*
- *the timestamp t of the last update, and*
- *reference P to parent Concept of dimensionality $l + 1$, if $l < d - 1$.*

The $d-l$ equations in Hessian normal form are the hyperplane equations that define the l-dimensional the subspace. These are obviously an essential part of the *Concept* as they are used for the unification with other *Concepts* and also are part of the final result of CashStream. The mean μ is the centroid of the data objects that are assigned to the corresponding cluster and is used for checking whether the *Concept* can be merged with another one. N denotes the number of data objects that are assigned to the cluster. This value and the timestamp t of the last update of this *Concept* are used to calculate an importance score for the *Concept*. The importance scores are used to weight the *Concepts* for the unification of two similar *Concepts*, since a recent *Concept* that represents a large number of data objects should contribute more than a stale *Concept* that does not represent as many objects. Finally, a *Concept* also includes a reference to a parent *Concept*, i.e., a *Concept* representing a higher-dimensional subspace in which the child *Concept* is embedded. This enables CashStream the retrieval of a cluster hierarchy.

On Representing Hyperplanes in Hessian Normal Form

The Hessian Normal Form (HNF) [226] has proven to be a well-suited representation for linear correlation cluster models. The main motivation behind using it as abstraction for linearly correlated subspace clusters for CashStream is that it contains a normal vector which describes the orientation of the corresponding hyperplane, respectively subspace. This is essential for our unification step as we use the orientations of two subspaces to determine their similarity. By using the HNF, we can formally describe a $(d-1)$-dimensional hyperplane $\mathcal{H}$ as

$$\vec{x} \cdot \vec{n} + b = 0,$$

with $\cdot$ indicating the scalar product, $\vec{x} \in \mathbb{R}^d$ denoting a data point lying on the hyperplane, $\vec{n} \in \mathbb{R}^d$ denoting the unit normal vector and b being the minimum distance between the hyperplane and the origin. Every data point $\vec{x}$ that solves this equation lies on $\mathcal{H}$. However, since subspace clusters typically are not perfectly correlated, we consider a data point $\vec{x} \in \mathbb{R}^d$ belonging to a subspace cluster of dimensionality $d-1$ whose hyperplane is defined by $\vec{n}$ and b, if it fulfills

$$\vec{x} \cdot \vec{n} + b \leq \epsilon.$$

Note that the ϵ parameter is a threshold parameter that is implicitly defined by setting the *maxSplit* parameter, i.e., the parameter that basically defines the size of a grid cell on the lowest split level.

As can be seen in Definition 9, a *Concept* contains $d-l$ of such hyperplane equations in HNF. This is due to the necessity of requiring $d-l$ HNF equation for describing a l-dimensional subspace. Intuitively, this can be understood as follows: if $d-l$ $(d-1)$-dimensional hyperplanes intersect in a d-dimensional space (with $l<d$), the intersection is a l-dimensional hyperplane. Mathematically, this can be seen as solving a simple linear system

$$Ax = b,$$

with A denoting an $m \times d$ matrix, where m is the number of normal vectors. If $d > m$, the linear system is underdetermined and hence the solution set describes a $(d-m)$-dimensional subspace.

As an example, consider Figure 5.3 where we are given a 3-dimensional data space with an 1-dimensional correlation cluster (red dots). In fact, the 1-dimensional cluster can be described by the intersection of two planes with collinear normal vectors that both contain all data points of the cluster. Therefore, a possible solution of CashStream in this case could be the following two equations (depicted as planes in Figure 5.3):

$$A: \; 0.41x + 0.41y - 0.82z + 0.82 = 0,$$
$$B: \; 0.86x + 0.05y - 0.73z + 0.73 = 0.$$

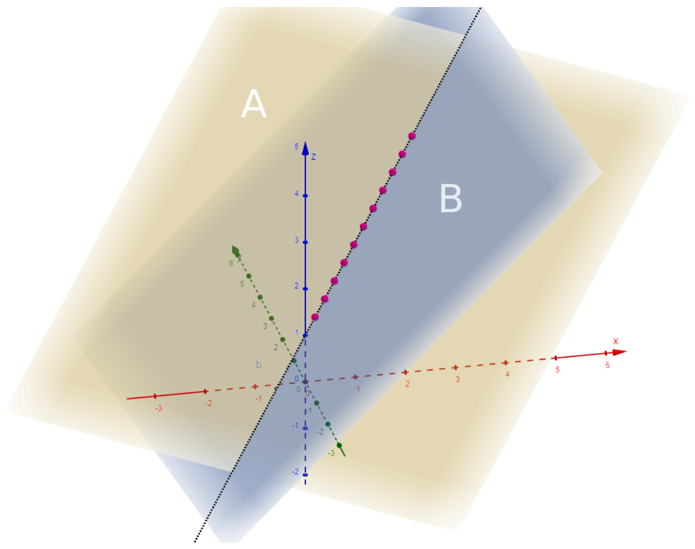


Figure 5.3: The intersection of two planes corresponds to the 1-dimensional correlation cluster

As described in Section 5.2, CashStream projects the data objects of an i-dimensional cluster onto the corresponding $(i-1)$-dimensional subspace to find even lower dimensional clusters. In particular, it also produces an i-dimensional normal vector $\vec{n}_i$ to define an i-dimensional basis B_i from which the $(i-1)$-dimensional subspace is derived as $B_i \setminus \vec{n}_i \in \mathbb{R}^{i-1}$ in this step. By doing this iteratively until no lower dimensional subspace can be found, the *CASH* procedure finally retrieves an ordered set of $d-l$ HNF equations[4] for a l-dimensional subspace, i.e.,

$$\begin{aligned}
\vec{n}_d \cdot x + r_0 &= 0 \\
\vec{n}_{d-1} \cdot (B_d \setminus \vec{n}_d \cdot x) + r_1 &= 0 \\
\vec{n}_{d-2} \cdot (B_{d-1} \setminus \vec{n}_{d-1} \cdot (B_d \setminus \vec{n}_d \cdot x)) + r_2 &= 0 \\
&\vdots
\end{aligned}$$

with $\vec{n}_{d-i} \in \mathbb{R}^{d-i}$, with $0 \leq i < l$, denoting the $(d-i)$-dimensional normal vector that defines the $(d-i)$-dimensional basis B_{d-i}, x being a data point associated with the i-dimensional subspace cluster and r_i being the distances between the subspace hyperplane and the origin. $B_{d-i} \setminus \vec{n}_{d-i}$ is a $(d-i-1) \times (d-i)$ projection matrix that is used to project $(d-i)$-dimensional data objects onto the $(d-i-1)$-dimensional subspace.

However, for measuring the similarity between two *Concepts*, CASH-STREAM requires each normal vector to be d-dimensional. We therefore reconstruct d-dimensional normal vectors from lower-dimensional normal vectors as follows. Let $\vec{n}_{d-i} \in \mathbb{R}^{d-i}$, with $0 < i < l$, be the $(d-i)$-dimensional normal vector defining the $(d-i-1)$-dimensional subspace whose basis is denoted as $B_{d-i-1} = B_{d-i} \setminus \vec{n}_{d-i}$, then the reconstructed d-dimensional normal vector $\vec{n}'_d \in \mathbb{R}^d$ is

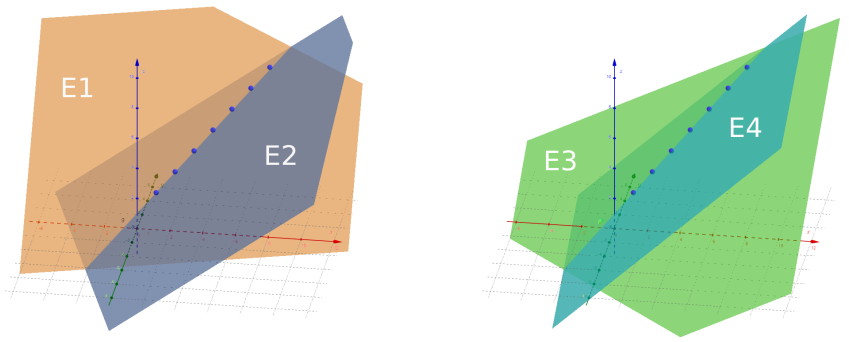
$$\vec{n}'_d = ((((\vec{n}_{d-i} \cdot B_{d-i+1} \setminus \vec{n}_{d-i+1}) \cdot B_{d-i+2} \setminus \vec{n}_{d-i+2}) \cdot \ldots) \cdot B_d \setminus \vec{n}_d).$$

Employing this reconstruction strategy to all $(d-i)$-dimensional normal vectors with $0 < i < l$ in addition with the d-dimensional normal vector $\vec{n}_d$ finally results in the desired set of $d-l$ non-parallel, and hence linearly independent [72], d-dimensional normal vectors that define the $d-l$ hyperplane equations of a *Concept*.

Similarity between *Concepts*

Theoretically, there is an infinite number of sets of equations describing a single subspace cluster, e.g., the 1-dimensional subspace cluster in Figure 5.3 since its model is the intersection of two planes. As shown in Figure 5.4, the straight line can be modeled by the intersection of two hyperplanes, the orientation of which is not necessarily important.

[4]The equations are in the ordering of the corresponding subspace dimensionality, i.e., the first equation defines the $(d-1)$-dimensional subspace, the second equation defines the $(d-2)$-dimensional subspace, and so on.



$E1: \; 0.26x - 0.80y + 0.53z = \;\; 0.52$
$E2: \; 0.58x + 0.19y - 0.78z = -0.79$

$E3: \; 0.41x + 0.41y - 0.82z = -0.82$
$E4: \; 0.68x + 0.05y - 0.73z = -0.73$

Figure 5.4: Sometimes the algorithm produces equivalent, but different models in terms of the model equations for the same linear correlation cluster, especially when finding the same subspace cluster within different batches.

In terms of *Concept* similarity, this means that two *Concepts* shall be considered similar as long as the intersections of their subspace equations describe approximately the same subspace, regardless the orientations of their subspace equations when considering them individually. Given this observation and the fact that each subspace hyperplane is defined by its normal vectors which are present in the HNF representations, we formalize the distance measure based on the following idea: Understanding an intersecting set of hyperplanes as the set of their respective normal vectors, every other normal vector contained in a second set of equations representing the same linear subspace is linearly dependent to the first set. Considering the example in Figure 5.4, this means that when regarding the normal vectors $\vec{n}_{E1}$ and $\vec{n}_{E2}$ of the hyperplane equations $E1$ and $E2$, and normal vectors $\vec{n}_{E3}$ and $\vec{n}_{E4}$ of the hyperplane equations $E3$ and $E4$, both $\{\vec{n}_{E1}, \vec{n}_{E2}, \vec{n}_{E3}\}$ and $\{\vec{n}_{E1}, \vec{n}_{E2}, \vec{n}_{E4}\}$ are sets of linearly dependent vectors. However, since we aim at measuring the similarity between two *Concepts*, we are interested in quantifying the linear dependence of these vectors rather than just determining whether they are linearly dependent or not. In fact, two *Concepts* are considered similar if their subspaces have a similar orientation, or in other words, if their normal vectors are approximately linear dependent. Therefore, the proposed similarity measure makes use of singular values.

Given a set of linearly independent normal vectors $V = \{\vec{n}_1, ..., \vec{n}_k\}$, we quantify the linear dependence of another vector $\vec{m}$ with respect to V by calculating the singular values $SV(A)$ of the matrix $A = (\vec{n_1}, ..., \vec{n_k}, \vec{m})$ and dividing the smallest value by the largest one. The closer the resulting value

$$L_{dep}\left(\underbrace{\vec{v}_1, ...\vec{v}_k, \vec{m}}_{\text{A}}\right) = \frac{\min\left(SV(A)\right)}{\max\left(SV(A)\right)}$$

is to zero, the closer the vectors of the matrix are to being linearly dependent through adding $\vec{m}$. Given two *Concepts* C_1 and C_2 with their sets of normal vectors N_1 and N_2 being of the same cardinality k, and each normal vector representing a $d-k$-dimensional linear subspace, we define the *Singular Value Distance* as follows:

$$SV_{dist}(C_1, C_2) = \max_{\vec{n} \in N_2}\left(L_{dep}(N_1, \vec{n})\right).$$

Note that this distance measure only accounts for the orientation of the correlation clusters described by the *Concepts*. In particular, two *Concepts* that describe different, parallel subspaces would have a singular value distance equal to zero. To avoid an unification of such *Concepts* we introduce a secondary measure accounting for the actual distance in an Euclidean sense between two *Concepts*. Precisely, we measure the Euclidean distance between two *Concepts* C_1 and C_2 by plugging in any data point lying on the subspace hyperplane of C_1 into the subspace equation of C_2. By doing this, we get the distance one has to shift the hyperplane of C_1 in direction of the normal vector such that the data point is contained in the plane. In other words, the perpendicular distance from the point to the plane is given by

$$d_{perp}(p, E) = |n_1 p_1 + ... + n_d p_d - r|$$

with p denoting the data point, E denoting the HNF equation and n being the corresponding normal vector. However, since the actual data points that defined a subspace are not available due to aggregating the necessary information within the *Concept* structure, we use the centroid of the *Concept* as representative data point. Thus, we compute the *Equation Shift Distance* between two *Concepts* C_1 and C_2 as:

$$d_{shift}(C_1, C_2) = \max_{i=1,...,k} d_{perp}(\vec{\mu}_2, E_{1_i}),$$

with $E_{1,i}$ being the hyperplane equations of C_1 and $\vec{\mu}_2$ being the mean of all data points forming the subspace captured in C_2.

In summary, we determine the similarity between a pair of *Concepts* by calculating the singular value distance that accounts for the orientation of the corresponding subspaces. If the two subspaces have a sufficiently low singular value distance, we calculate the perpendicular distance between the two subspaces to exclude parallel but distant subspaces from being unified. If the perpendicular distance is below a certain threshold, the two subspaces are considered to be similar enough such that the two corresponding *Concepts* are unified according to the unification procedure described in the following subsection.

Aging and Unification

Informally, the unification of two *Concepts* is the process of merging two subspace cluster representatives. This can be done quite efficiently due to *Concepts* being data structures whose entries either can just be overwritten (in case of timestamp or pointer to the parent *Concept*) or are additive (in case of hyperplane equations, number of absorbed data objects or mean).

Aging

However, when unifying two *Concepts* it is important to consider the importance of the *Concepts*, as for instance a very recent *Concept* is typically more important than a stale *Concept*, or a *Concept* which represents lots of data objects is more important than a *Concept* that represents only a few. Therefore, we introduce an *importance score* for each *Concept* which we use as weighting factor when merging two *Concepts*. Beside that, the temporal part of the importance score is also used to discard very old *Concepts* that are considered irrelevant for an up-to-date subspace clustering model. Formally, we define the importance score of a *Concept* C as

$$\mathcal{I}(C) = e^{-\lambda \Delta t} \cdot N_C$$

with λ being the decay parameter, Δt being the temporal difference between the current timestamp and the timestamp given in C and N_C being the number of data objects that have been assigned to C. The first part of this equation, i.e., $e^{-\lambda \Delta t}$ is referred to as temporal part and contains the damping factor $\lambda > 0$. A high value of λ means low importance of old data and vice versa. However, to save space and computational costs a *Concept* should be pruned if it remains in memory unchanged for a large amount of time, resp. a large amount of batches. We therefore introduce a threshold parameter θ which basically models a sliding window approach as a *Concept* whose temporal part of the importance score falls below the threshold θ is discarded. Note that the size of the sliding window depends on θ. In our experiments, we mostly used $\lambda = 0.2$ and we discarded *Concepts* from memory that had an importance score below $\theta = 0.05$. The resulting temporal decay function is depicted in Figure 5.5.

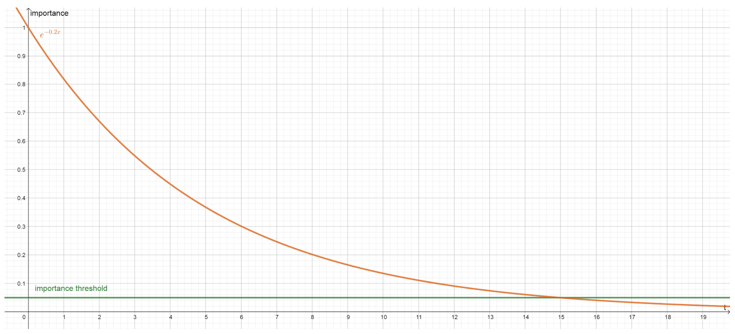


Figure 5.5: Graph of the importance score for $\theta = 0.2$. The x-axis denotes the time, the y-axis denotes the importance score.

Unification

After extracting the new *Concepts* of a batch with *CASH* and recalculating the importance score of all *Concepts* in memory, we perform a unification step of the new *Concepts* and the *Concepts* currently in the memory. Beginning at dimensionality $d-1$, we compare the current *Concepts* with the new ones in terms of similarity and unify two *Concepts* if they are similar enough with respect to some similarity threshold. The unification is continued in descending order regarding dimensionality. If two *Concepts* C_1 (current) and C_2 (new) of the same dimensionality have been confirmed as similar enough, the following operations are performed to create the resulting *Concept* C^*[5]:

- For each pair of equations $E_{1,i}$ and $E_{2,i}$ with $0 < i < d-l$, we define a new Equation E_i^* by using the weighted mean of the normal vectors and the weighted mean of the distances to the origin of the two equations. At this, the weight of the new *Concept* is N_{C_2}, the old is weighted by its importance score, such that

$$E_i^* = \frac{\mathcal{I}(C_1) \cdot n_{E_{1,i}} + \mathcal{I}(C_2) \cdot n_{E_{2,i}}}{2} \cdot x + \frac{\mathcal{I}(C_1) \cdot r_{E_{1,i}} + \mathcal{I}(C_2) \cdot r_{E_{2,i}}}{2}.$$

  This way a new and possibly slightly shifted set of hyperplane equations is created.

[5]Note that we assume one of the *Concepts* to be a novel *Concept* without loss of generality. The only difference between merging a stale *Concept* with a new one is that the temporal part of the importance score function becomes 1 for newly created *Concepts*, while it is less than 1 for older ones.

- The mean representative of all data points from C^* is calculated by weighting the respective means from C_1 and C_2 by their importance, i.e.,

$$\mu_{C^*} = \frac{\mathcal{I}(C_1) \cdot \mu_{C_1} + \mathcal{I}(C_2) \cdot \mu_{C_2}}{2}.$$

- The number of data objects represented by C^* is the sum of data objects represented by C_1 and C_2, i.e., $N_{C^*} = N_{C_1} + N_{C_2}$

- The timestamp of C^* is set to the current timestamp, i.e., the timestamp of newly created *Concept* C_1, such that $t_{C^*} = t_{C_1}$.

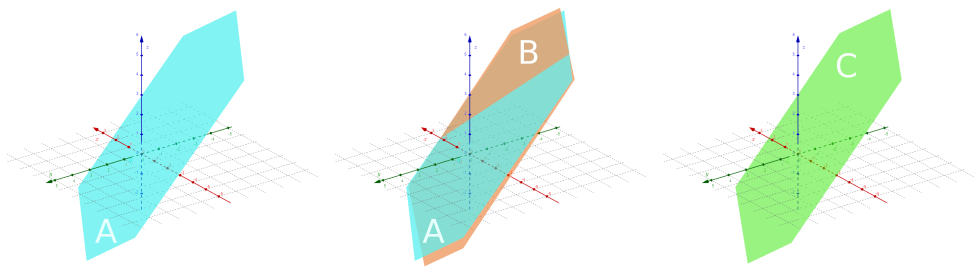
- The reference to the parent *Concept* of C^* will be set to the parent *Concept* of C_1. If there is any child *Concept* with either C_1 or C_2 as their parent, its parent pointer will be set to C^*.

Algorithm 4 *unifyConcepts*

```
Input: Set of Concepts $\mathcal{C}$, current timestamp $t_{curr}$, damping factor $\lambda$, temporal
    threshold $\theta$, singular value distance threshold $\tau_{SVdist}$, equation shift distance
    threshold $\tau_{ESdist}$
 1: for each Concept $c$ having $t = t_{curr}$ do
 2:   for each Concept $c'$ having $t \neq t_{curr}$ do
 3:     if $e^{-\lambda \cdot (t_{curr} - t)} < \theta$ then
 4:       remove $c'$ from $\mathcal{C}$                    // remove stale Concepts
 5:       continue
 6:     end if
 7:     if $|E_c| == |E_{c'}|$ then
 8:       $dist_{SV} = SV_{dist}(c, c')$
 9:       if $dist_{SV} \leq \tau_{SVdist}$ then
10:         $dist_{shift} = d_{shift}(c, c')$
11:         if $dist_{shift} \leq \tau_{ESdist}$ then
12:           c $\leftarrow$ unify $c$ and $c'$ as described above
13:           remove $c'$ from $\mathcal{C}$
14:         end if
15:       end if
16:     end if
17:   end for
18: end for
```

Tracking Concept Drifts

Since *Concepts* do not have to be completely equal with respect to normal vectors and origin distances in order to trigger the unification, there will be



$A:\ 0.41x + 0.66y + 0.62z = 0.3$ $B:\ 0.43x + 0.64y + 0.64z = 0.38$ $C:\ 0.42x + 0.65y + 0.63z = 0.34$

Figure 5.6: Visual example for the unification of *Concepts* A and B. The resulting *Concept* is denoted as C in the right-far figure. For ease of presentation, A and B are weighted equally.

some movement of the yet found subspace clusters. In some applications it might be useful to record these movements, e.g., to detect abnormal behaviors, or on a slower scale to track effects reasoned by wearing in a machine monitoring application. In fact, every *Concept* that results from an unification can be seen as a weighted compromise between the older and the more important newer *Concept*. Usually, there are two possible scenarios which we refer to as stabilization and concept drift. Stabilization means there are several similar *Concepts* from succeeding batches that stand for a linear correlation that does not change over time, the movements mostly cancel each other out. Concept Drifts indicate a rotation or parallel shift of one or several plane equations describing the *Concept*. For every equation in the *Concept* resulting from an unification of an old and a new *Concept*, we record the difference of the normal vectors between the equations and between the distances to the origin. When a *Concept* is extracted from a batch, these difference vectors (movements) are initialized as zero-vectors. For every unification, the actual movements are added to them. For the stabilization scenario, the movement vectors of a *Concept* will be close to zero vectors, even after several unifications. For the drift scenario, we can observe what kind of movement has been made by looking at the movement vectors. For instance, the unification done as seen in Figure 5.6 would yield a movement of

$$\vec{v_0} = (0.1, 0.1, 0.1);\ d = 0.04$$

This corresponds to a rotation of the old *Concept* A by about 1° and an origin distance by 0.02, and a later unification would simply be added to this movement. However, this comes to the costs of requiring additional memory space for the movement vectors and the single floating point numbers that describe the *Concepts*' drifts of the distances to the origin.

Complexity

To conclude this Section, we briefly take a note on the complexity of CashStream. As shown in [3], the average time complexity for the static *CASH* procedure is

$$O(s \cdot c \cdot N \cdot d^3),$$

with s denoting the *maxSplits* parameter, c being the expected number of clusters, N denoting the number of data points and d denoting the dimensionality of the data set. Due to the batch-wise processing scheme in *CashStream*, the average time complexity for the generation of all *Concepts* can be reformulated as

$$O(s \cdot c \cdot k \cdot B \cdot d^3),$$

where s, c and d again denote the *maxSplits*, expected number of clusters and dimensionality, and k is the number of batches and B is the batch size, i.e., $k = N/B$. Additionally, CashStream requires the unification of *Concepts* after the processing of each batch. This procedure requires pairwise distance calculations between the *Concepts* and each distance calculation (note that the singular value distance is the dominating part) is in $O((d-l) \cdot d(d-l+1)^2)$, with l denoting the dimensionality of the subspace[6], due to performing SVD decompositions to get the singular values. In total, the worst case amount of distance calculations is $c \cdot (c-1)$ per batch, given that c is the expected number of clusters and all clusters have the same subspace dimensionality. Considering all k batches, the worst case complexity (under the assumption that the expected number of clusters is c) is $O(k \cdot c(c-1) \cdot (d-1) \cdot d^3)$, if the subspace clusters are all distinct and one-dimensional. Therefore, the overall complexity of CashStream is

$$O(s \cdot c \cdot k \cdot B \cdot d^3 + k \cdot c(c-1) \cdot (d-1) \cdot d^3).$$

However, we want to emphasize that the latter part of the term (the complexity over all unification steps) is almost non-relevant in practice as the worst case scenario is usually very unlikely. In fact, our experimentations show that the latter part nearly has no influence at all when considering the runtime. Also, we want to note that the complexity in terms of memory usage can be decreased, and most importantly, the batch-wise processing scheme enables to run the Hough-based subspace clustering procedure on commodity hardware, even for very large databases.

[6]Recall that the number of subspace equations is d minus the dimensionality of the subspace l, which leads to a $d \times (d-l+1)$ matrix that needs to be decomposed ($+1$ due to appending one normal vector of the second *Concept* to test linear dependency, the factor $(d-l)$ because we need to check this for each normal vector).

5.4 Experiments

In this section, we evaluate CashStream by comparing the proposed streaming algorithm against the static counterpart *CASH*. Precisely, we evaluate the performance with respect to accuracy, memory consumption and finally the throughput. All these measures are important metrics for streaming methods as such algorithms typically trade some accuracy for a drastically decreased memory consumption, and the throughput finally measures how many data objects can be processed within a certain amount of time.

Datasets

We use synthetic and real world datasets throughout this section. Generally, the synthetic datasets are generated such that there are subspace clusters within a higher-dimensional data space. The clusters can have a pre-defined amount of jitter, i.e., some deviation from being perfectly correlated, and we also generate noise which are data points that do not belong to any cluster. The real-world dataset is a slightly manipulated version of the *wages dataset*. The original version of the dataset[7] has also been used in [3], and consists of 534 records each having four different features, i.e., age, years of education, years of experience and salary. However, since the original version only counts 534 records, we enlarge it by copying and shuffling the records such that we finally have 20000 data points. This way, we can use the data to simulate a data stream appropriately.

Parameter Settings

If not stated differently, we perform grid searches over various parameter settings and report the results for the best settings. Precisely, we range the parameters over the following sets:

- damping factor $\lambda \in \{0.2, 0.5, 0.8\}$
- temporal threshold $\theta \in \{0.2, 0.5, 0.8, 1.\}$
- singular value distance threshold $\tau_{SVdist} \in \{0.005, 0.01, 0.02, 0.03\}$
- equation shift distance threshold $\tau_{ESdist} \in \{0.05, 0.1, 0.15\}$

Note that we use the number of a batch as timestamp for that batch, i.e., timestamp $i \in \mathbb{N}$ is assigned to the the *Concepts* extracted from the i-th batch. The min. points parameter *minPts* that must be set for *CASH* is set

[7]`http://lib.stat.cmu.edu/datasets/CPS_85_Wages`

Batch size	Num. of batches	ARI	AMI	maxSplits	minPoint fraction
1000	1	0.9991	0.9937	10	0.75
500	2	0.9991	0.9937	10	0.75
333	3	0.9909	0.9603	10	0.75
200	5	0.9982	0.9889	10	0.75

Table 5.1: Results for CashStream on the simple line dataset.

proportionally to the batch size. We fix the *minPts fraction* $\tilde{m}$ to a value of 0.75, resulting in *minPts* parameters being $\tilde{m} \cdot k$ with k being the batch size. The other *CASH* specific parameter *maxSplits* is set according to the data set at hand and reported for each experiment individually.

Clustering Quality

For measuring the clustering quality of StreamCash, we compare the resulting clusterings for several different settings of the batch size parameter, including the batch size for which a single batch contains the entire data set, which is equivalent to the static *CASH*. We employ the following two quality measures:

- *ARI*, the Adjusted Rand Index [126], which is an extension of the Rand Index (*RI*) [216] that accounts for chance, meaning that the expected value of the ARI for completely random partition labeling is 0. We use it to compare the clustering result retrieved by *CashStream* to the ground truth of the synthetic data.
- *AMI*, the Adjusted Mutual Information [258] is also adjusted for chance and is another measure for similarity of clusterings.

Note that due to the lack of ground truth in the real-world dataset, we restrict ourselves to synthetic datasets in the experiments evaluating the clustering quality. Our first dataset contains a simple 1-dimensional line in a 3-dimensional data space. The set contains of 1000 data points of which 200 are random noise points, and 800 belong to the cluster. The line does not contain any jitter (thus simulating a perfectly correlated data set), and there is no hyperplane of points containing the line.
In Table 5.1 we report the best calculated *ARI* and *AMI* scores that we achieved when running CashStream for various parameter settings on the simple line data set.

The results show that there is almost no loss in terms of clustering quality when splitting the dataset into batches. This is the expected result if there

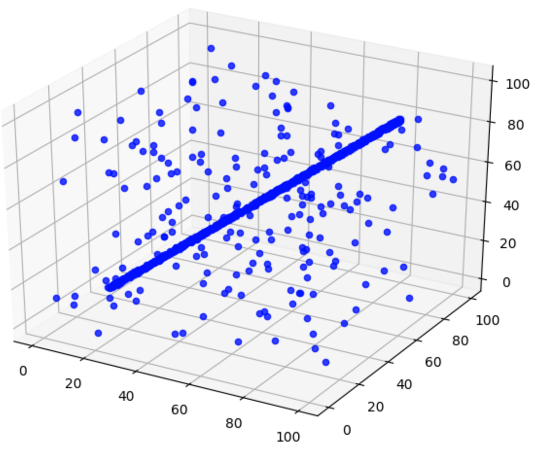


Figure 5.7: Visualization of the Simple Line data set with 20% random noise

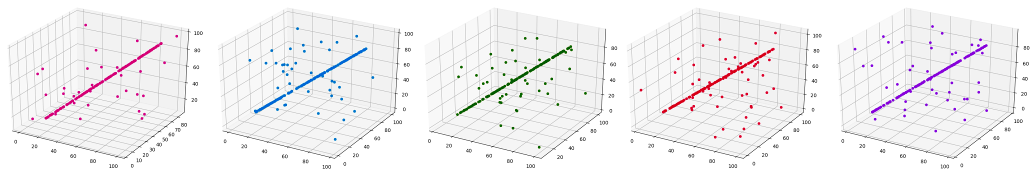


Figure 5.8: The Simple Line dataset split into 5 batches and visualized.

is no difference in the data distributions over the different batches. A visualization of the distributions within the used batches can be seen in Figure 5.9. We therefore conclude that the batch-wise processing scheme is indeed a well-suited approach for stretching the workload of *CASH*.

The second dataset in this chapter is a 4-dimensional set of points containing two 2-dimensional planes of 1000 data points each and 1000 random noise points, thus 3000 points in total. The planes both are jittered, making the data not perfectly correlated within their corresponding subspaces. This way, we can evaluate how well the algorithm can deal with imperfect data (as it might appear in real world applications). For a better understanding, the amount of jitter applied to the planes is visualized for a $2-$dimensional plane in a $3-$dimensional space in Figure 5.9.

The calculated ARI and AMI for this dataset can be seen in Table 5.4. Note that we slightly adopted the *maxSplits* and the *minPts fraction* parameters compared to the previous experiment, as we also used larger batch sizes. In general, it can be observed that the clustering quality slightly drops when choosing a batch size below 1000. This might indicate that the subsample might not reflect the data distribution sufficiently when choosing the

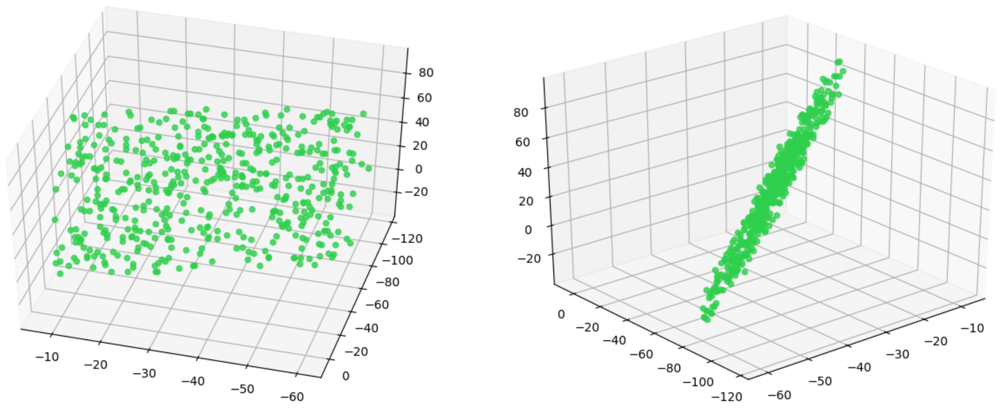


Figure 5.9: Two different perspectives on a 2−dimensional plane in a 3−dimensional space generated with our plane generator, where the maximal jitter distance of plane points is an euclidean distance 5.0 as for the Two Planes dataset.

Batch Size	Batches	ARI	AMI	maxSplits	minPts fraction
3000	1	0.951	0.922	9	0.3
1500	2	0.943	0.907	9	0.3
1000	3	0.924	0.881	9	0.3
750	4	0.875	0.829	9	0.3

Table 5.2: Results for CashStream on the perfect planes dataset.

batch size too small, which can be especially problematic in scenarios where correlations are imperfect. Another reason for the decreasing clustering accuracy can be the presence of temporal effects (i.e., slight drifts in the data distribution, increasing amount of noise, etc.).

Throughput

In this section, we investigate the actual throughput in terms of data points per second of the algorithm. In general, our evaluation of the throughput can be understood as a runtime comparison between the batched algorithm and the static *CASH*. For the throughput experiment, we used two synthetic and one real-world dataset of various sizes and dimensionality to demonstrate the scalability of the batched streaming approach. For each dataset we report the throughput in data points per second and the total runtime in seconds. Each of the reported values is the mean value over three runs. For all those runs, we compared (by means of comparing the detected subspaces) the re-

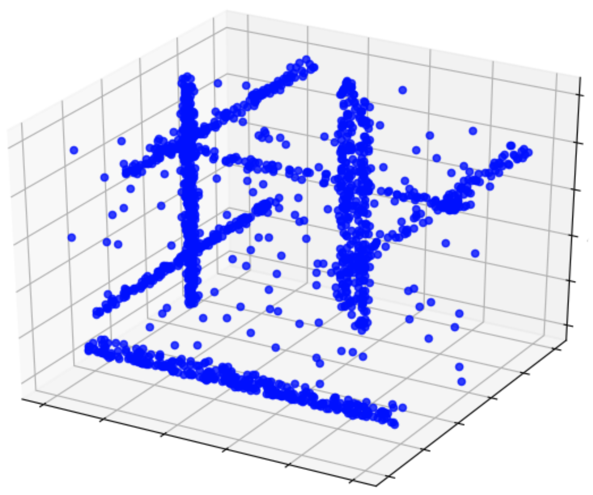


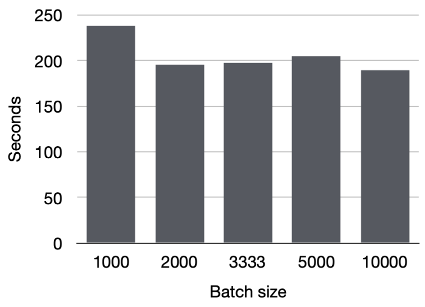
Figure 5.10: The multi correlation set is our first throughput experiment set. It contains several highly jittered, 1-dimensional correlation clusters which form several 2-dimensional hyperplanes throughout the data space.

sulting clustering models with the clustering models that can be expected and selected the parameter settings according to the best result.

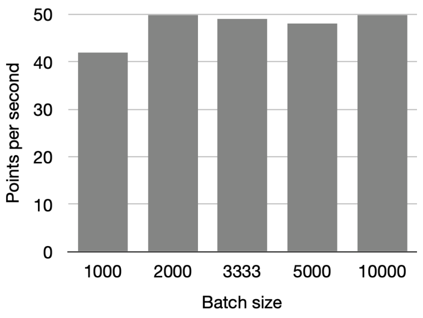
Perfect Planes set. Firstly, we use a synthetic dataset which we refer to as *perfect planes dataset*. It consists of 10000, 5-dimensional data points which form two perfectly correlated 3-dimensional linear correlation clusters of 4500 points each and 1000 random noise points. By using this dataset, we study the performance of our algorithm under perfect conditions.

The results, depicted in Figure 5.11, confirm our expectations from Section 5.3 as there is no notable relationship between runtime and batch size. In general, the runtime of the batched CashStream algorithm is comparable to the runtime needed for the static *CASH* algorithm (the two right-most bars).

Multi Correlation set. As second dataset we use a 3-dimensional data set that contains several highly jittered linear correlations of varying size and 10% random noise. The set consists of 20000 points and can be seen in Figure 5.10. We refer to this dataset as *multi correlation dataset*. This dataset is somewhat more difficult due to data points belonging to the same subspace cluster are not correlated perfectly. Also, it appears that some of the clusters form a hierarchy, i.e., multiple one-dimensional subspace clusters may form a two-dimensional subspace cluster. We compare the throughput and runtime

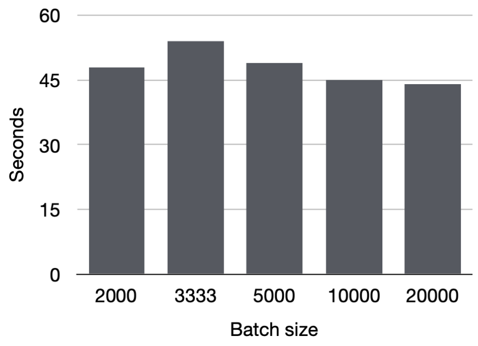


(a) Runtime

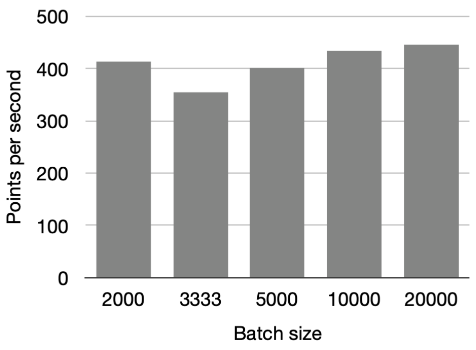


(b) Throughput

Figure 5.11: Throughput/Runtime experiment for the Perfect planes dataset evaluated for several different batch sizes, *maxSplits*=7, *minPts fraction*=0.42

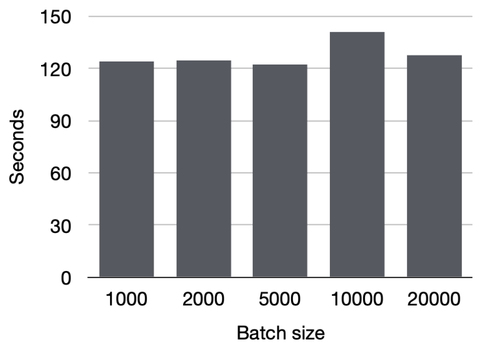


(a) Runtime

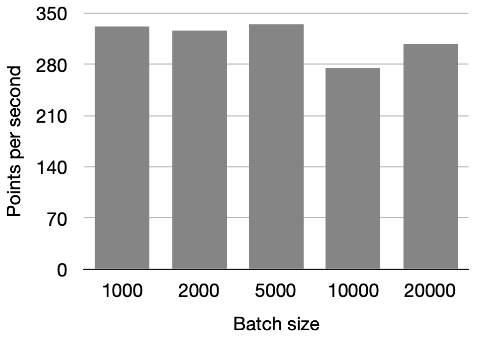


(b) Throughput

Figure 5.12: Throughput/Runtime experiment for the Multi Correlation dataset evaluated for several different batch sizes, *maxSplits*=8, *minPts fraction*=0.2



(a) Runtime



(b) Throughput

Figure 5.13: Throughput/Runtime experiment for the enlarged wages dataset evaluated for several different batch sizes, *maxSplits*=8, *minPts fraction*=0.13

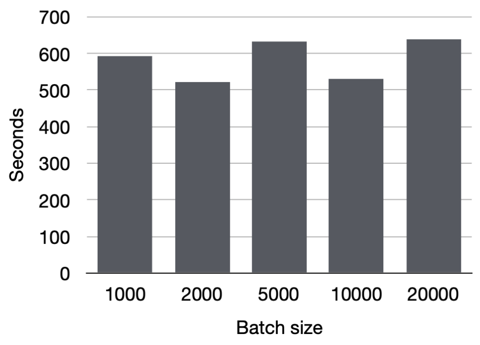
for different batch sizes, summarized in Figure 5.12.

As suspected in the complexity discussion above, the batch size does not seem to have a large impact on the total computation time for this data set. We can see the largest runtime deviation for a batch size of 3333, which is most likely to be explained by a “bad batch”, i.e., a batch for which the *minPts* parameter is slightly too low, which in turn leads to the situation where CashStream needs to evaluate noisy *Concepts*.

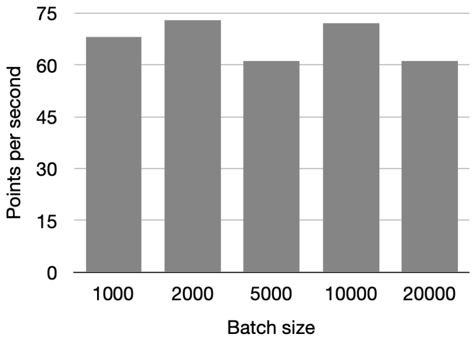
Enlarged Wages dataset. Finally, we use the slightly manipulated version of the *wages dataset* as a real-world dataset. Recall that the original version of this dataset consists of 534 records each having four different features, i.e., age, years of education, years of experience and salary. We enlarged the original version by copying and shuffling the records such that we finally have 20000 data points so that we can use the data to simulate a data stream appropriately. In this experiment, we once again compare the throughput (and runtime) for several batch sizes including the full data set.

As can be seen in Figure 5.13, this experiment once more supports our assumption from above, suggesting there is no real dependency between runtime and batch size in practice.

To investigate the influence of the split parameter on the runtime, we increased the split level to 10 and repeated the experiment. As can be seen in Figure 5.14, the total amount of computation time increases (as *CASH* must process more cuboids in total), but the relation between runtime and batch size remains approximately the same when varying the batch size.



(a) Runtime



(b) Throughput

Figure 5.14: Throughput/Runtime experiment for the enlarged wages dataset evaluated for several different batch sizes, *maxSplits*=10, *minPts fraction*=0.13

Memory

As memory consumption is a critical metric in terms of streaming applications, we show the monitored RAM usage of the batched approach and compare it with the static *CASH* in this section. Precisely, we demonstrate that the memory peaks achieved by the batched approach stay close to a constant level, no matter how many points are being processed. This is due to the fact that we only keep the light-weight abstractions of the clustering models in memory throughout the execution. As a general comparison, we take a look at the heap space usage of the respective approaches, as the heap space is the central indicator of how much memory an application needs at runtime[8]. Intuitively, we expect the following heap space profile for the approaches. For the static approach, we expect the heap space usage sloping up steadily throughout the execution time, since the amount of cuboids in the queue is growing. In the batched approach, we expect the heap space usage to reset to almost none after processing a batch, since between batches the only thing to keep in memory are minimal clustering models. We sketched that expectation in Figure 5.15. The graphs shown in this section were created using Java ViusalVM 1.4.2, which is included in the Java JDK. To force the JVM to be more efficient with RAM and to simulate a light-weight system, we capped the maximal available heap space to around 2GB.

Enlarged Wages Dataset. For this experiment, we again use the enlarged wages dataset as in Section 5.4. We showcase three different batch

[8]It is noteworthy that the code is implemented in Java

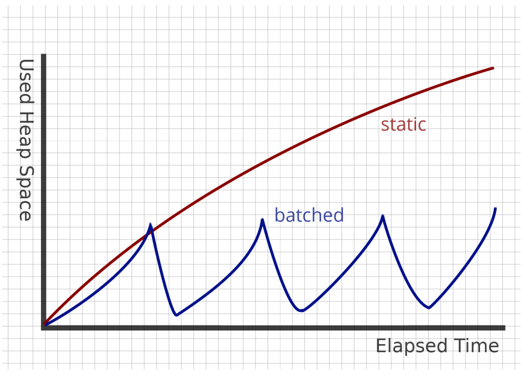


Figure 5.15: A sketch of the expected heap space usage comparing the methods

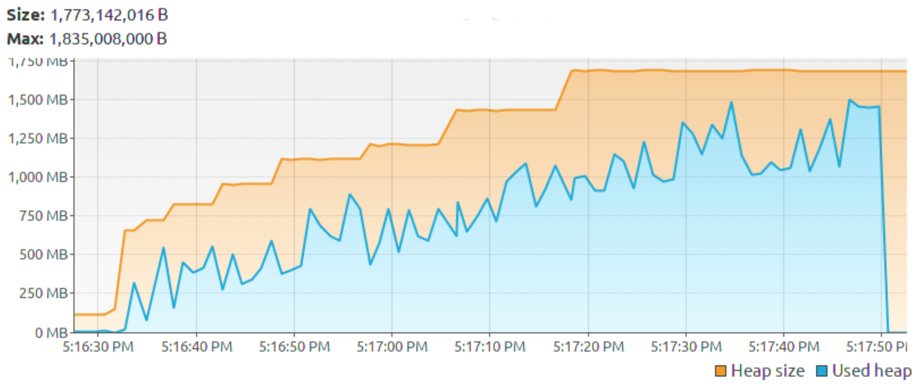


Figure 5.16: The Heap Space usage profile for the enlarged wages dataset in the static approach, processing one batch of size 20000 with *maxSplits*=8, *minPts fraction*=0.2

sizes, 20000, 6666 and 2000 with constant *maxSplits* of 8 and and a *minPts fraction* of 0.2.
The static approach simulated in the full-sized 20000 points batch behaves much like expected, with the heap space rising steadily to a maximum level of around 1500MB.

When subdividing the points into 3 batches of 6666 points, we observe two crucial details: Firstly, the peak Heap Space usage is at around 850MB, which is significantly lower than in the static approach. Secondly, the three sequentially processed batches can clearly be identified as three peaks in the Heap Space profile. Another thing to note is that the three batches can vary both in the time used for processing them and in the Heap Space used.

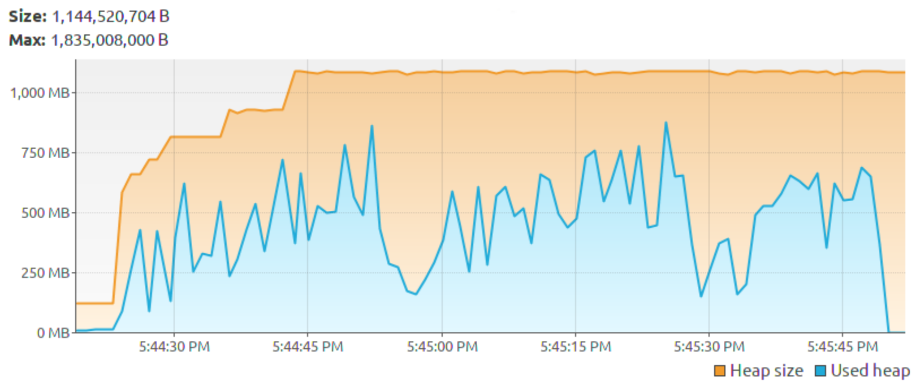


Figure 5.17: The Heap Space usage profile for the enlarged wages dataset in the batched approach, processing three batches of size 6666 with *maxSplits*=8, *minPts fraction*=0.2

This is heavily dependent on how "good" the data points are arranged for *CASH*, i.e., there is not much noise close to actual clusters filling up the division queue, the *minPts* parameter is not set too low etc. Finally, when subdividing the data into 10 batches, we see that the peaks of RAM usage are close to being constant at 400MB, with rare deviations to higher values. After close consideration one can still make out batch-like structures in the Heap Space profile. [9]

Multi Correlation set. We also performed the same experiment on the multi correlation set, for which we can make the same observations. The results for a batch size of 20000 and for a batch size of 1000 are depicted in Figures 5.19.

Again, for the static approach we see one peak that reaches almost 1200MB. Using a smaller batch size, i.e., 1000, again seems very economical with regards to memory usage as we have around 300-400MB memory consumption on average with very few peaks going up to 500MB.

In conclusion, we can say that a batch size of around 1000-2000 is a very good way to control the memory usage of the algorithm. We have also seen that the larger a data set is, the higher the memory usage rises in the static approach. The streaming approach CASHSTREAM generally is on a much lower base level of RAM usage, with the RAM usage dropping to quite low values as soon as a batch is compressed into *Concept* structures.

[9]Note that the RAM usage is very low compared to the static approach. In fact the dips could just be regular garbage collections in the middle of single a batch processing.

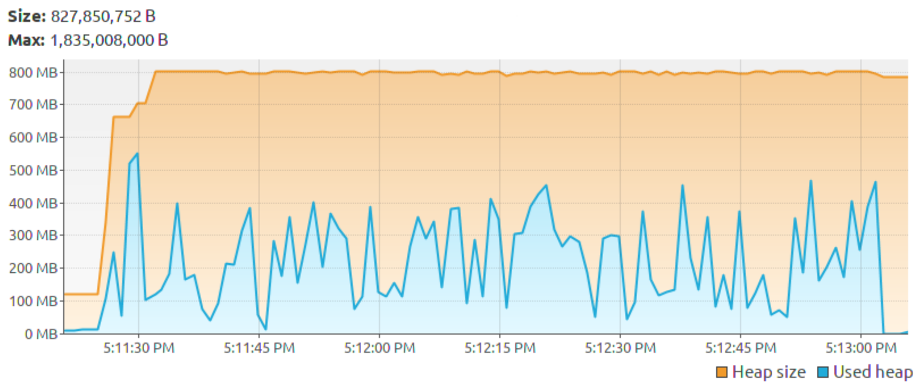
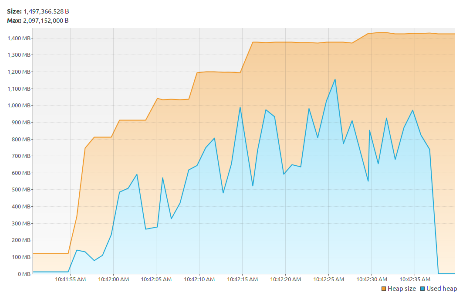
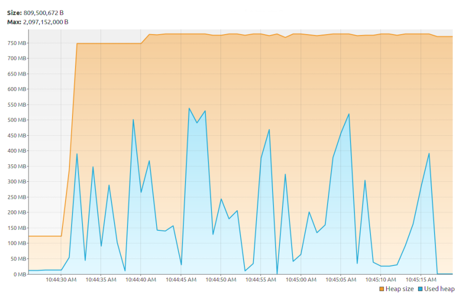


Figure 5.18: The Heap Space usage profile for the enlarged wages dataset in the batched approach, processing ten batches of size 2000 with *maxSplits*=8, *minPts fraction*=0.2



(a) Static



(b) Batched

Figure 5.19: The Heap Space usage profile for the multi correlation dataset in the static approach, and the batched version with *maxSplits*=8, *minPts fraction*=0.2

5.5 Conclusion

In this chapter, we presented the novel subspace clustering algorithm CashStream that is able to deal with high-dimensional streaming data efficiently. Precisely, CashStream relies on the subspace clustering paradigm that was introduced for the static *CASH* algorithm, i.e., using Hough transformations to identify interesting linear subspaces. However, in contrast to *CASH*, the proposed algorithm uses a batch processing scheme, identifies interesting subspaces within the data batches, and subsequently compresses important information within *Concept* data structures that have been designed such that they fulfill the necessary properties for streaming applications, i.e., additivity and incrementality. This way, CashStream does not need to keep all data points in memory and hence requires much less memory than its static counterpart. The additivity property allows to keep the number of different *Concepts* small as similar *Concepts* can be merged efficiently. Due to the incrementality property of *Concepts*, the algorithm can make use of an aging mechanism to downgrade stale *Concepts* such that most recent data coming from the data stream is considered more important than stale data. We propose a damping window approach to downgrade old *Concepts* gradually and a sliding window alike mechanism to get rid of *Concepts* whose importance falls below a certain threshold. This keeps the amount of data kept in memory small. Our experimental evaluation showed that CashStream is fairly robust against different choices for the batch size and simultaneously reduces the memory consumption significantly compared to the static *CASH* algorithm (less than 50% on the real-world dataset). At the same time the loss in terms of clustering quality is negligible. To further boost the performance, an interesting direction for future work is to scale the computation of subspace clusters horizontally and distribute cluster calculations over multiple machines. Furthermore, it might be worth to investigate other distance measures for the merging criterion as SVD might become computationally expensive for very high-dimensional subspaces.

Chapter 6

User Identification by Using Microblog Data

The work presented in this chapter has been published as the article *On privacy in spatio-temporal data: User Identification by Using Microblog Data* in the Proceedings of the International Symposium on Spatial and Temporal Databases, 2017 [228].

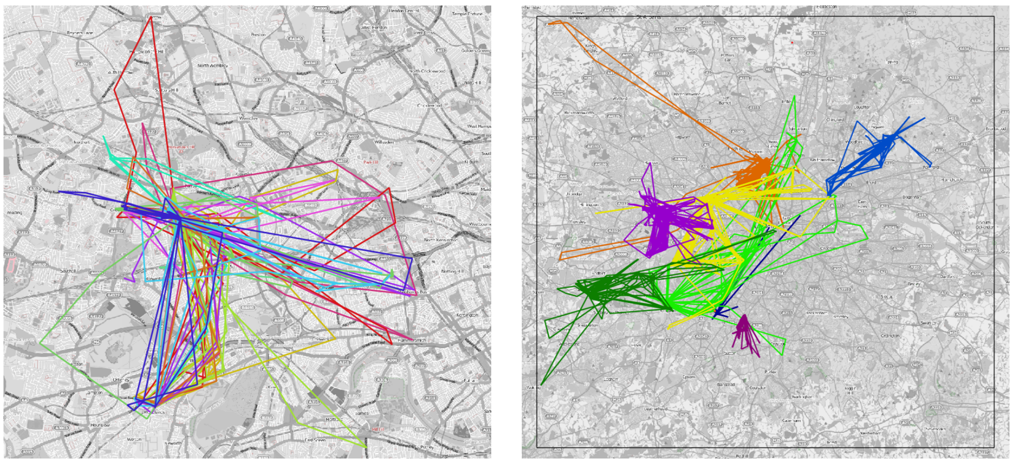
6.1 Introduction

Nowadays, billions of copies of applications distributed by Apple's App Store® access a user's geographic location. As an example, Niantic's well known augmented reality game "Pokémon Go", which has been downloaded more than 100 million times on Android devices alone [1], constantly synchronizes the GPS location of users with a company server. While users trust that their location data will be used in sensitive fashion, Apple® for instance collects, uses and shares "precise location data, including the real-time geographic location" of their customers' devices with "partners and licensees" [2].

The mobility behavior of a person often reveals a large variety of sensitive information, which they may not be aware of. A list of potentially sensitive professional and personal information that could be inferred about an individual, knowing only their mobility trace, was published recently by the Electronic Frontier Foundation [40]. Such personal information could simply be marketing information, obtained from a user's choice of restaurants,

[1] `https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo` [date: 2019-03-21]

[2] According to the Apple Privacy Policy from May 22, 2018. `https://www.apple.com/legal/privacy/en-ww/` [date: 2019-03-21]



(a) Weekly history of a single user.   (b) 12-week trace of 10 users.

Figure 6.1: Illustration of Twitter Traces

or a user's religious beliefs, inferred through the proximity to a particular church. It can also indicate other, much more sensitive, information about an individual based on their presence in a motel or at a medical clinic.

In this chapter, the severity of privacy risks through publishing individual spatio-temporal data on the use case of Twitter data is investigated. In particular, it is shown that geotagged tweets might yield enough location information for building user specific trace profiles. Based on these profiles, Twitter accounts can be linked to additional trace data being observed from unknown users. Other location based services or mobile devices are also potential sources for traces. Additionally, face detection methods tag known persons in images in social networks. Thus, geotagged images can reveal a user's whereabouts at certain points in time. Given that there are multiple such images, it might be possible to build a trace and link it to a known user. To conclude, freely available location data might be used to link accounts and devices for the same user. Thus, the user reveals more of their movements and actions than might be intended.

To derive trace profiles for a given Twitter account, geotagged tweets containing an exact geolocation, a time, and a user ID were collected. Since this work focuses on the location aspect the content of the Tweet is completely ignored, even though it might add even more useful information to user profile. Using the Twitter API, or similar micro-blogging applications, users can publish a short text message, called a Tweet, together with their current geolocation, a current time-stamp, and their user ID.

The sequence of Tweets of a user is interpreted as a trace. For each user, all available Twitter data is used to build a trace profile to capture each user's specific mobility patterns. Using these profiles, new trace, for which the originating user is unknown, can be linked to a known user with an alarmingly high accuracy. To illustrate this classification problem, a typical Twitter trace of a single user is depicted in Figure 6.1(a). The figure shows a twelve week trace of a user's tweets, in color-coded one-week intervals. For comparison, Figure 6.1(b) shows the same twelve week traces for ten users, using a different color per user. Note that the tweets of this user are voluntarily published by the user, such that Figure 6.1(a) and Figure 6.1(b) do not raise any privacy concerns.

The challenge of this study is to match a new trace, such as a one week trace corresponding to a single color in Figure 6.1(a), to the correct user corresponding to one of the colors in Figure 6.1(b). Note that the ten selected user profiles in the example are located in relatively distinct activity regions. Thus, finding the right profile is relatively simple. In a more realistic setting, distinguishing thousands of users in the same area, and user identification is significantly more challenging. In these experiments up to 15,989 users, within the same bounding box of London, are used leading to a much more challenging classification task.

Twitter data is comparatively sparse to other location tracking applications, as tweets are typically published at a frequency of less than one per hour. Despite this data sparsity, it is shown that a large quantity of low-quality location data can still be used to construct highly discriminative user models. To summarize the contributions of the work presented in this chapter are as follows:

- Trace models to capture user-specific movement profiles from sparse traces obtained from Twitter.
- Methods for mapping a newly observed traces of an unknown user to the most likely user in the database.
- An experimental evaluation showing that individual patterns are highly unique and allow for a user classification accuracy of up to 98% .
- A case study of linking users of Twitter to users of Instagram, with an accuracy of up to 81%.

The remainder of this paper is organized as follows. Chapter 6.2 briefly positions the work with respect to related works, and subsequently the problem setting is formalized and the task of linking new traces to users is defined.

Chapter 6.3 describes the trace models and the approach to user identification. The results of the experimental evaluation are described in Chapter 6.4. Scalability of this solution is address in Chapter 6.4 and further user linkage experiments are address in Chapter 6.4. It is concluded in Chapter 6.5.

6.2 Problem Definition

In general, user identification is focused on identifying the same user again within the same database, while user linkage is focused on linking two users together across multiple sources of data. Note that the work presented in the remainder of this chapter assumes user trace data to be fully available, without any notion of privacy preservation. This assumption is appropriate in the experimental evaluation, using publicly available Twitter data. However, other datasets may employ some form of privacy, thus it is important to understand privacy methods, which might be used on this data. We therefore survey related state-of-the-art methods for user identification, user linkage, mobility patterns and spatial privacy in Section 3.3.

In this work, the question of to what extent a set of spatio-temporal observations, such as geotagged tweets, are sufficient to derive spatial user profiles for the observed users and reliably link location traces of unknown users to one of the known user profiles is answered. Therefore, this section will define terms and notations, and formally define the problem of user identification using trace data.

In this paper, spatio-temporal data is considered. That is data of users annotated with a geolocation and a timestamp, such as obtained from Twitter.

Definition 10 (Spatio-Temporal Database)**.** *Let $\mathcal{U}$ denote a set of unique user identifiers, let $\mathcal{S}$ be a set of spatial regions, and let $\mathcal{T}$ denote a time domain. A* spatio-temporal database *$\mathcal{DB} \subseteq \mathcal{U} \times \mathcal{S} \times \mathcal{T}$ is a collection of triples $(id \in \mathcal{U}, s \in \mathcal{S}, t \in \mathcal{T})$. Each triple $(u, s, t) \in \mathcal{DB}$ is called an observation.*

Furthermore, a trajectory is defined as a sequence of location and time pairs.

Definition 11 (Trajectory)**.** *A trajectory $tr \subseteq \mathcal{S} \times \mathcal{T}$ is a collection of pairs $(s \in \mathcal{S}, t \in \mathcal{T})$.*

These trajectories are then partitioned temporally and spatially to distill the information down to a minimum set of components.

To build a user specific mobility pattern, the data is temporally partitioned into equal sized time intervals called epochs. Within an epoch the set of observations of a specific user is called a location trace, formally defined as follows.

Definition 12 (Location Trace). *Let $\mathcal{DB}$ be a spatio-temporal database. Let $\mathcal{E} = \{e_1, ...e_n\}$ be a partitioning of $\mathcal{T}$ into n temporal intervals denoted as epochs. For each epoch $e \in \mathcal{E}$, and each user u, the trace*

$$\mathcal{DB}(u, e) := \{(u', s, t) \in \mathcal{DB} | u' = u, t \in e\}, \tag{6.1}$$

is called the location trace of user id during epoch e.

In the remainder of this paper, location trace models are introduced to capture the motion of a user in space and time. The models are derived from the set of all trace of a user, called their trace profile formally defined as follows.

Definition 13 (Trace Profile). *Let $\mathcal{DB}$ be a spatio-temporal database, let $u \in \mathcal{U}$ be a user and let $\mathcal{E}$ be a temporal partitioning of $\mathcal{DB}$ into n epochs. The trace profile $\mathcal{P}(u)$ is the set of all traces of u, i.e.,*

$$\mathcal{P}(u) = \{D(u', e) | u' = u, e \in \mathcal{E}\}. \tag{6.2}$$

This trace profile is used to establish a pattern between multiple traces over discrete epochs, in order to allow for more accurate user identification.

The main challenge of this paper is to map the trace of an unknown user to a user already in the database, thus identifying them.

Definition 14 (User Identification). *Let $\mathcal{DB}$ be a spatio-temporal database and let $Q \subseteq \mathcal{S} \times \mathcal{T}$ be a trace of an unknown user u. The task of user identification is to predict the identity of user u of Q given $\mathcal{DB}$. The function*

$$\mathcal{I} : \mathcal{P}(\mathcal{S} \times \mathcal{T}) \mapsto \mathcal{U}, \tag{6.3}$$

maps a trace Q to user x as a user identification function.

Thus, user identification is a classification task mapping a trace to its unknown user. This is not to be confused with a de-anoymization attack, such as in [37] where a user's real world identity is uncovered. This task is only attempting to identify which user's Trace Profile is most similar to the new trace. To train a user identification function $I(Q)$, the next chapter presents the classification approach, which uses the traces in $\mathcal{DB}$ as a training set, in order to predict the user of a new trace $Q \notin \mathcal{DB}$.

User linkage, takes this task further and attempts to map two users, from separate databases, together and is formally defined as follows.

Definition 15 (User Linkage). *Let $\mathcal{DB}_1$ and $\mathcal{DB}_2$ be two trace feature databases, and let $\mathcal{U}_{\mathcal{DB}_1}$ and $\mathcal{U}_{\mathcal{DB}_1}$ be two user databases. Such that each entry $(T, u) \in \mathcal{DB}_1$ corresponds to a trace feature vector $\mathcal{T}$ and a user $u \in \mathcal{U}_{\mathcal{DB}_1}$, and each $(T', u') \in \mathcal{DB}_2$ corresponds to a trace feature vector $\mathcal{T}'$ and a user $u' \in \mathcal{U}_{\mathcal{DB}_2}$. The task of user linkage is to map a user in $\mathcal{U}_{\mathcal{DB}_1}$ to a user in $\mathcal{U}_{\mathcal{DB}_2}$.*

This is achieved by using the user identification techniques discussed in the next chapter and will be discussed in Chapter 6.4.

6.3 Trajectory based User Identification

Trace models are introduced to capture the motion of a user $u \in \mathcal{U}$ in space and time by learning from their trace profile $\mathcal{P}(u)$ in Subsection 6.3. Note that this first approach does not consider the time component of observations of a user within an epoch. The time component is only used to divide the whole trajectory of a user into different epochs that can be used for learning and testing. For each model, a similarity measure to quantify similarity between different trace models is proposed. Based on these similarity measures, the user identification approach is presented in Subsection 6.3. As mentioned before, the prediction is based on the assumption that there exists a profile $P(u_i)$ for each user $u_i \in U$.

Trace Profile Modeling

Each trace $\mathcal{DB}(u, e)$ of user u during epoch e is a sequence of observations, i.e., time-stamped geo-locations. A spatial grid to partition geo-space into equal sized regions $\mathcal{S} = \{S_1, S_{|S|}\}$ is used, thus reducing a trace to a sequence of time-stamped grid-cells. To model such a sequence, two kinds of approaches are proposed:

- The first approach using *set descriptors* treats a trace as a *set* of grid-cell observations, thus ignoring the sequence, ordering, and time-stamps of these observations.
- The second approach using *frequent transitions* considers the transitions of users from one spatial region to another, thus explicitly modeling the order of observations.

Set Descriptors

Ignoring the temporal aspect, a trace $\mathcal{DB}(u, e)$ of user u during epoch e can be described by a vector $v(u, e)$ of all spatial regions in S. In other words, each spatial region is represented by a dimension of $v(u, e)$.

Note that $v(u, e)$ contains zero values in the majority of dimensions as each user usually only traverses a small fraction of space during an epoch. In other words, $v(u, e)$ is sparse. Modeling trace using frequency descriptions has a strong resemblance to handling bag of words vectors known in text mining. To describe, if and how often a domain was visited within trace $\mathcal{DB}(u, e)$, the following two approaches are examined.

Binary Descriptor In this rather simple method, a trace $\mathcal{DB}(u, e)$ is represented as a set of visited spatial regions. Thus, each feature value v^{bit} equals one if user u visited region S_i (at least once) during epoch e, formally:

$$v_i^{\text{bit}}(u, e) := \begin{cases} 1, & \text{if } \exists (u', s, t) \in \mathcal{DB} : u' = u \wedge s \in S_i \wedge t \in e, \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

To compare binary vectors $v, v' \in \{0, 1\}^n$, the Jaccard coefficient is employed [128], which is a standard similarity measure for sets:

Definition 16 (Jaccard Coefficient)**.** *Let $v, v' \in \{0, 1\}^n$ be two bit vectors, then the Jaccard coefficient is defined as follows:*

$$Jac(v, v') = \frac{\sum_{i=1}^{n} v_i \wedge v'_i}{\sum_{i=1}^{n} v_i \vee v'_i} \tag{6.5}$$

Frequency Descriptors A frequency, or term weighted, vector [222] v^{freq} contains the number of visits of each spatial region of user u in epoch e. This allows to distinguish between users visiting a particular region more or less often than other users.

$$v^{\text{freq}}(u, e)_i = |\{(u', s, t) \in \mathcal{DB} | u' = u \wedge s \in S_i \wedge t \in e\}|. \tag{6.6}$$

A common way to compute the similarity in sparse numerical vectors is the cosine coefficient:

Definition 17 (Cosine Coefficient)**.** *Let $v, v' \in \mathbb{N}^n$ be two vectors, then the Cosine coefficient is defined as follows:*

$$Cos(v, v') = \frac{v \cdot v'}{||v|| \cdot ||v'||} \tag{6.7}$$

Since the cosine coefficient can be strongly dominated by dimensions having high average frequency values, spatial regions are normalized by their total number of observations [222].

Transition Descriptors

All of the previous trace descriptors had in common that they treat a trace as an unordered set of locations, without considering any notion of sequence or time. In this section, a trace is treated as a sequence of regions. As a baseline to compute the similarity between two sequences, dynamic time-warping [35] (DTW), a state-of-the-art method for similarity search on sequences, is used. Since the experimental evaluation shows that using DTW without any adaption as a similarity measure yields a fairly low classification accuracy, this section presents two approaches to directly model the transitions of a trace. A transition is a pair (s, s') of regions where s is called source and s' is called destination. Using a descriptor for each pair of spatial regions s_i, s_j, describing the number of times the specific sequence (s_i, s_j) has been observed in a trace $\mathcal{DB}(u, e)$, is proposed.

Definition 18 (Trace Transitions). *Let $\mathcal{DB}(u, e) = \{(s_1, t_1), ..., (s_n, t_n))\}$ be a trace, the set of n transitions $\uparrow \mathcal{DB}(u, e)$ is defined as the multi-set (thus allowing duplicates)*

$$\uparrow \mathcal{DB}(u, e) := \bigvee_{1 \leq i < n} (s_i, s_{i+1}). \tag{6.8}$$

The number of occurrences of (s, s') in trace $\mathcal{DB}(s, e)$ is denoted as $\uparrow \mathcal{DB}(u, e)(s, s')$.

Since modeling all observed transitions blows up the feature space quadratically, Using only the k globally most frequent transitions as features is proposed.

- **Frequent Transitions:** The globally most frequent transitions are searched for and the number of occurrences of these transitions is used as a feature vector to describe a trace.
- **Transition Probabilities:** Common transitions of two traces are found, and their similarities are adapted by the global rarity of these transitions.

Definition 19 (Top-k Most Frequent Transitions). *Let k be a positive integer, then the set FT is a set of pairs of spatial regions defined as*

$$FT^k(\mathcal{DB}) = argmax^k_{s_i, s_j \in \mathcal{S}} |\{ \sum_{u \in \mathcal{U}, e \in \mathcal{E}} \uparrow \mathcal{DB}(u, e)(s_i, s_j)\}|, \tag{6.9}$$

where $argmax^k_X(\varphi)$ returns the set of k arguments $x \in X$ yielding the maximum value substituted in term φ.

Now the k most frequent transitions $FT^k(\mathcal{DB})$ can be used as additional features. Similar to the set descriptors presented in Subsection 6.3, the features are described using

- Bit vectors, using the feature vector

$$v_i^{\uparrow \text{bit}(u,e)} = \begin{cases} 1 & \text{if } FT^k(\mathcal{DB})_i \in \uparrow \mathcal{DB}(u,e) \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

- Frequency vectors, using the binary feature vector

$$v^{\uparrow \text{freq}}(u,e)_i = \uparrow \mathcal{DB}(u,e)(FT^k(\mathcal{DB})_i) \tag{6.11}$$

For these vectors, the same similarity functions defined in Section 6.3 can be used.

Classification

Regardless of which of the modeling approaches presented in this section is employed, the result is a high-dimensional feature vector. To classify a new trace of an unknown user, the next section proposes the classification procedure, using the previously proposed user-specific trace models. To classify the user of a new trace, a k-nearest neighbor classification approach is employed. This choice is made due to the extremely high dimensional feature space, having one dimension per spatial grid-cell. Therefore, given a trace database $\mathcal{DB}$, traces $\mathcal{DB}(u,e)$ are extracted for each user u in each epoch e. Since the user is known for each of these traces, the result is a labeled dataset P_{train} of feature vectors. Given a new trace Q, map Q to its feature description v_{new} and search the k-nearest neighbors of v_{new} in P_{train} w.r.t. a corresponding similarity measure. To decide the final class decision, each queried neighbor is weighted by its similarity value and the class is predicted as the one having the largest cumulated similarity.

Formally, the k-nearest neighbors classification can be defined as follows. Let $P_{train} = \{(v_i, y_i) \mid v_i \in \{0,1\}^n \wedge y_i \in \mathcal{L}\}$ be the set of training instances consisting of pairs (v_i, y_i) with v_i being the feature description of the user trace i and y_i being the label, i.e., identity of the user, assigned to trace i. $\mathcal{L}$ denotes the set of labels. Given the feature description v_{new} of a query trace, the identity, resp. label, y_{new} of v_{new} is determined by cumulating the similarities, i.e., $d(.,.)$, for each label $l \in \mathcal{L}$ represented among the k-nearest neighbors of v_{new} and taking the most representative label.

$$y_{new} = argmax_{l \in \mathcal{L}} \{ \sum d(v_{new}, v_k^l) \mid v_k^l \in kNN(v_{new}) \} \tag{6.12}$$

Note that no index structure is used to support the kNN-search due to the high dimensionality of the feature space.

User Linkage

In addition to the identification of individual users, another application of the user trace profiling is to link users between two trace datasets. Therefore, let $\mathcal{DB}$ and $\mathcal{DB}'$ be two trace databases having the set of users $\mathcal{U}$ and $\mathcal{U}'$, respectively. The task of user linkage is to find pairs of database users $(u \in \mathcal{U}, u' \in \mathcal{U}')$ that correspond to the same individual in the real world, i.e., having $u = u'$. As an example, the two datasets may correspond to Twitter and Instagram. The same individual may have different user names in both social networks. The task of user linkage is to find such individuals.

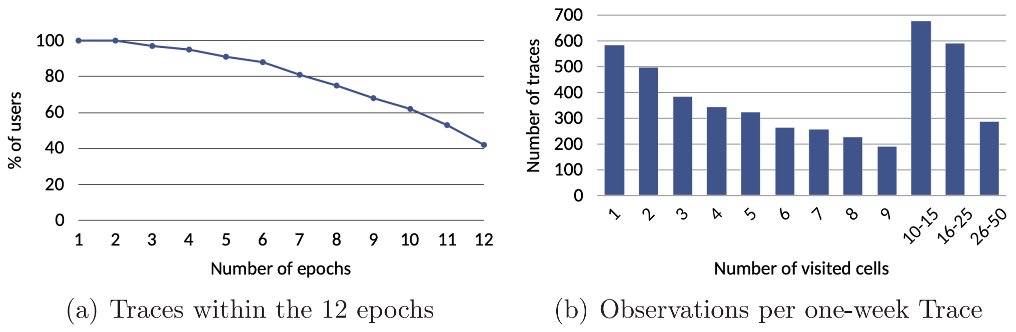
Clearly, using the approach presented in Section 6.3, the trace of each user are classified in $\mathcal{DB}$, and the most similar user in $\mathcal{DB}'$ is classified. The drawback of such approach is that multiple users in $\mathcal{DB}$ may be matched to the same user in $\mathcal{DB}'$, and some users in $\mathcal{DB}\prime$ might not have any match. To avoid this drawback, the matching problem is formalized as a bipartite graph, containing for each $(u \in \mathcal{U}, u' \in \mathcal{U}')$ a weight of similarity. This similarity is chosen by performing a kNN search of each trace in $\mathcal{DB}$ on the database $\mathcal{DB}'$. Then, the score of (u, u') corresponds to the number of occurrences of $u\prime$ in kNN sets of all traces of user u.

Given this bipartite graph, the Hopcroft-Karp algorithm [119] is used to find an optimal matching, i.e., mapping of each user in the smaller database to exactly one user in the other that maximizes the total score.

6.4 Experimental Evaluation

The proposed approach is initially evaluated on a dataset mined from Twitter using their public API, feeding from a global 1%-sample using a $(51.25, 51.75)$ degrees longitude to $(-0.55, 0.30)$ degrees latitude window covering the London region shown in Figure 6.1(b). London was chosen as a starting location for having a high population, while still being predominantly an English speaking location. Furthermore, London shows a high density of users and tweets, thus increasing the size of the trace database $\mathcal{DB}$, and allowing more significant conclusions to be made.

It was also decided to use one-week periods for the temporal epoch. This choice was meant to minimize the daily variability in a users locations. For example: a user may work Monday through Friday, but only go to the gym Monday, Wednesday and Friday, or night classes on Tuesday and Thursday.



(a) Traces within the 12 epochs (b) Observations per one-week Trace

Figure 6.2: Distribution of the Top 500 most prolific users in our London-Twitter dataset.

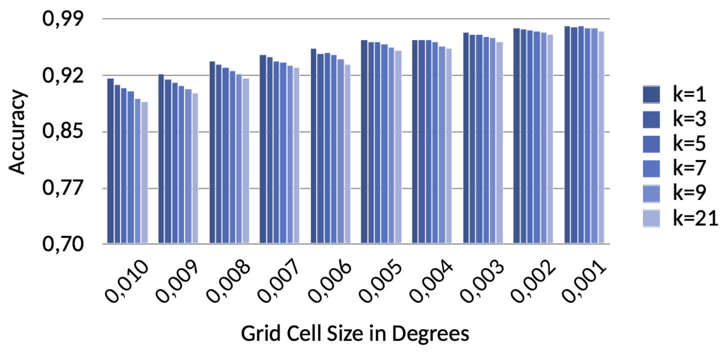
They may also go grocery shopping, or to a religious institution only on the weekends, thus going locations significantly different than were they would be on a week day. Additionally, Twitter data is extremely sparse for most users, thus more than a day was necessary to a reasonable number of tweets to create location traces from. This lead to the choice of a twelve week time interval from December 30, 2013 to March 24, 2014 being used. The choice of twelve weeks was to allow for multiple weeks of traces for each user, while not having too large of a dataset to use for initial testing purposes.

Out of these London-Tweets, the 500 users with the most Tweets during the study period were selected, excluding obvious spammer or bot users. This dataset was then split into temporal epochs of one-week. Thus, the database contains a total of $|\mathcal{U}| = 500$ users, and a total of $|\mathcal{E}| = 12$ epochs. Consequently, the database $\mathcal{DB}$ contains a total of $\mathcal{U} \times \mathcal{E} = 6000$ location traces.

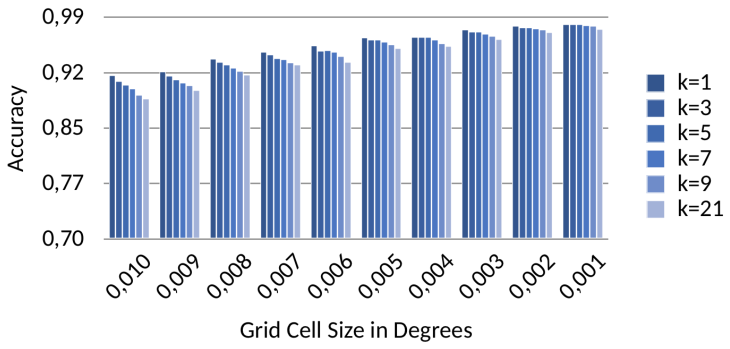
To discretize space, a spatial grid is applied on the aforementioned rectangle covering the London region, having an extent ext in longitude and latitude ranging from $0.01'$ to $0.001'$. The set of all resulting grid cells constitutes the set of spatial regions $\mathcal{S}$, having $|\mathcal{S}| = 4,250$ cells for $ext = 0.01'$ and $425,000$ cells for $ext = 0.001'$.

Consequently, for a user $u \in \mathcal{U}$ and an epoch $e \in \mathcal{E}$ a trace $\mathcal{DB}(u, e)$ is a sequence of cells in $\mathcal{S}$. To give a more detailed intuition of the characteristics of the dataset, Figure 6.2 shows statistics about the traces of these 500 users. Figure 6.2(a) shows the number of traces having at least one observation in the corresponding epoch.

Of users, 42% have an observation have at least one observation in each of the twelve epochs, and 75% of the users have at least one observation in at least eight epochs. In addition, Figure 6.2(b) shows the number of observed cells for each trace. Most users only visited a small number of space cells



(a) Jaccard Similarity



(b) Cosine Similarity

Figure 6.3: Classification Accuracy for varying grid-cell size and varying k.

each week, as half of the trace contain six or less cells. Note that any trace having zero observations were removed from the dataset.

The classification experiments in this work were performed using an eight-fold cross validation. Eight folds for optimal parallelization on an eight core processor. Thus, in each experiment a test set of tracestrace $Q(u, e) \subset \mathcal{DB}(u, e)$ is selected, and user mobility profiles are built using the techniques of Section 6.3, without using the test traces, i.e. $\mathcal{DB}(u, e) \backslash Q(u, e)$, in the training step to avoid over-fitting.

Note that this important avoidance of over-fitting is a main differentiation to the trace identification approach proposed in [77]. By having the query trace in the training data, a $k = 1$-NN classification would always return a 100% classification accuracy, but defeating the purpose of user identification. Consequently, since the related work in [77], solves a different problem, a comparison would be unfair and non-explanatory. See Section 3.3 for more details on [77].

As a classifier, k-nearest neighbor classification was utilized, using a distance-weighting in case of ties, which is able to perform well despite an extremely large number of $|\mathcal{S}|$ features. Classifications are performed using scikit-learn, a Python machine learning framework [207]. An exhaustive search of all combinations available in scikit-learn in order to determine the best possible settings to use. See Appendix A for raw results.

Accuracy Using Set Descriptors

In the first set of experiments, the accuracy of the user identification is evaluated for different grid-resolutions ext, using binary descriptors for the Jaccard similarity measure (c.f. Definition 6.3). The results of this evaluation are shown in Figure 6.3(a). In the basic setting having a relatively coarse spatial grid of $ext = 0.01'$, a simple distance weighted kNN classification is able to correctly identify (c.f. Definition 14) up to 85% of individuals for $k = 5$. This result improves even further as the grid-resolution ext is increased. In the case of the most detailed grid having $ext = 0.001'$, the solution is able to break the 97% classification accuracy line. This result is quite concerning, as it shows that the motion of individual real-persons is quite characteristic, and that the motion model allows to capture this individuality and allows to discriminate different users very well.

The classification result are worse for $k = 1$ and $k = 3$. This result is contributed to chance, as another user may, by chance, have a trace very similar to the query trace $q_u^e \in Q(u, e)$ of user u. However, by using more neighbors, it is likely that the correct user u appears at least twice in the $k = 3$ or $k = 5$ set, thus out-weighting the erroneous user in the first rank. Yet, for $k > 5$ there is a drop in accuracy. This is contributed that the query user only has at most 11 traces in the training set. This number might be less than 11 if a user was not active in all epochs. This is the case for many users, shown by Figure 6.2. In the extreme case having $k = 21$, at least 10 trace of wrong users must be in the kNN result, allowing noise have a much greater effect, especially in the case where u has few trace.

Furthermore, Figure 6.3(b) shows the results using frequency vectors as descriptors, and using the cosine coefficient as a similarity measure (c.f. Definition 6.3). The improvement in classification accuracy is relatively minor, but are able to hit the 98% accuracy mark. This result can be contributed to the fact that binary descriptors already perform so well. Summarizing, knowing the set of places that a user visited is descriptive enough, such that the frequency of visits does not yield much additional descriptiveness.

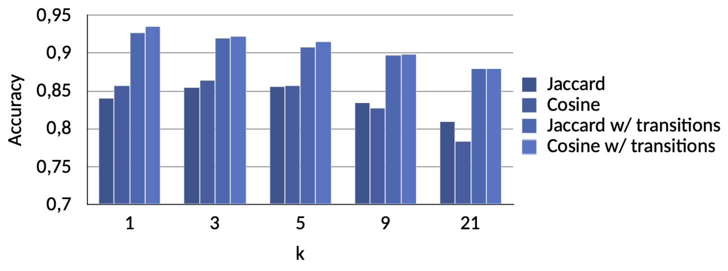


Figure 6.4: Classification Accuracy using Frequent Transitions.

Accuracy Using Frequent Transitions

In the next set of experiments, how the usage of transition descriptors (c.f. Section 6.3) instead of set descriptors affects the classification accuracy is evaluated. The results depicted in Figure 6.4 indicate that using from-to-transitions, as opposed to just using sets of cells, further allows to improve the classification quality. An increase in classification accuracy of around 10% (absolute) is observed using transitions, achieving an classification accuracy of nearly 95%. This result indicates that the sequence, and thus the motion in space and time is more descriptive than just sets of regions, and thus the motion in space-only.

While this was method did allow for a slight increase in accuracy, this increase came at a cost. It causes the dimensionality of the data, and greatly increase complexity. And thus, it greatly increases the processing requirements. Because there was only a small increase in accuracy for this increase in complexity, transitions were not use used in the remainder of the experiments. Though, further research in the subject could be worthwhile.

Accuracy for Different Observation Counts

Next, the number of observations required to identify (c.f. Definition 14) a user accurately is evaluated. Therefore counts are created according to the observation distribution in Figure 6.2. Then tests are for each count. If a trace does not have the minimum number of observations for the corresponding group, it is not tested, and if a trace has more observations than the allowed maximum for the corresponding group, a random sample is taken and tested instead. Thus, instead of testing the accuracy on the original traces this tests the accuracy on controlled observation counts.

The classification results for each group can be seen in Figure 6.5. Surprisingly, in the case of having only one random observation for each trace, it is possible to identify over 70% of the users in this dataset. This is likely due

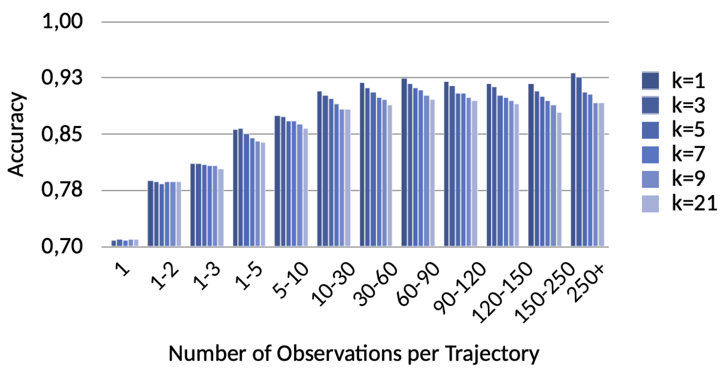


Figure 6.5: User identification accuracy for different observation counts.

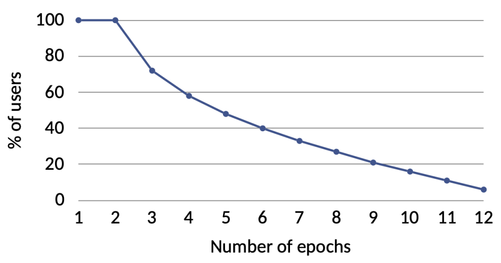
to the fact that a random location from a trace is likely to pick a users most frequent grid cell, which is most discriminative. Increasing the number of observation samples to two, a significant increase in accuracy to 78% is seen, and a steady growth in accuracy from there is shown. Accuracy starts to level off after having 30 or more obervations from a user. This is surprising, as the vast majority of trace has more than 60 observations. Thus, sampling down to 30 observations, yields a significant reduction in data, but as Figure 6.5 shows, yields almost no reduction of discriminative information.

The leveled accuracy level is above 90%, which is extremely high for a classification task having 500 different classes. This positive result is also a consequence of large trace (i.e., traces having a large number of observations) generally having larger trace in the training set, as the frequency distribution of tweets among these 500 Twitter users in London is very skewed. Finally, the classification performs the best, if the parameter of the kNN classification is set to $k = 1$. This result is in line with Figure 6.3(b), as Cosine-Similarity is used per default in this experiment.

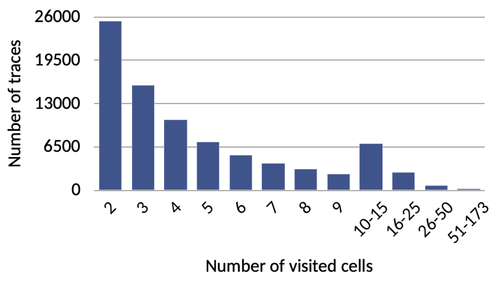
Summarizing this experiment, very short trace having 10 or less observations in space and time are enough to unveil the identity of a user. This is a concerning result.

User Linkage Between Different Social Networks

In all the previous experiments, a single user had to be identified based on a new trace. In this section, the next step is evaluated. Linking whole sets of users of two different social networks, based on their traces, as described in Section 6.3 and defined in Definition 15. For this purpose, two new datasets are employed, one generated synthetically by splitting the scalability (c.f. Chapter 6.4) dataset randomly, and one splitting the same dataset based on

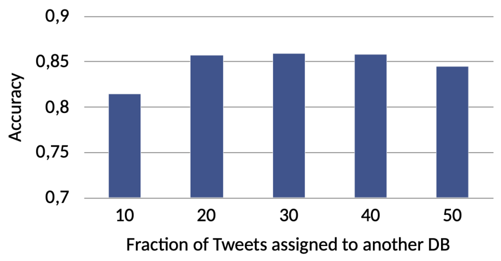


(a) Traces per user.

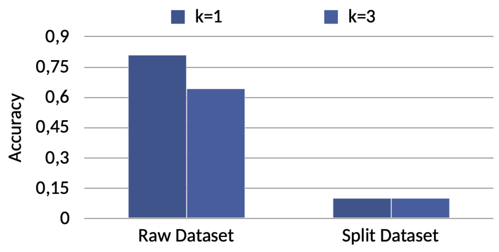


(b) Observations per one-week Trace

Figure 6.6: Distribution of all 15,989 users in our London-Twitter dataset.



(a) User Linkage results for different fractions of user belonging to each database.



(b) User Linkage results for linking Twitter and Instagram.

Figure 6.7: Classification Accuracy for different Social Networks.

links between Twitter and Instagram.

Synthetic Database Split: For the synthetic database, a fraction of p Tweets is uniformly sampled from the Twitter dataset $\mathcal{DB}$, and pretend that this set belongs to a different social network $\mathcal{DB}'$. In this sampled database $\mathcal{DB}'$, the user-labels as ground-truth, which the algorithm tries to predict given the data in $\mathcal{DB}$ can be used. For this experiment, only traces having at least 10 tweets to sample from are considered. If uniform sampling of a trace yields an empty set, it is re-sampled.

Instagram Data: Out of the 2.7 million tweets in the dataset, a significant portion of 204 thousand tweets is labelled as coming from the Instagram network. These Tweets were cross posted by the user, on both Instagram and Twitter. Thus, the Instagram database $\mathcal{DB}^I$ consists of all these cross-linked posts. For the Twitter database, two cases are evaluated. In the first case, the full dataset $\mathcal{DB}$ can simply be used, thus assuming that the Instagram observations were made in both datasets. In the second case, the database $\mathcal{DB}_T = \mathcal{DB} \setminus \mathcal{DB}_I$ is used, thus assuming that the Instagram observations were made in the Instagram network only.

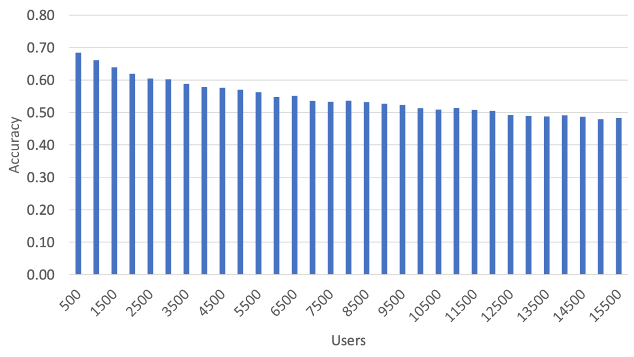
The results on the synthetic database split are shown in Figure 6.7(a). For each value of p, 10 random samples of the database $\mathcal{DB}$ are obtained, and results from each are averaged in order to avoid effects generated due to random sampled. In all ten runs, the depicted values showed almost no deviation, all being in a $\pm 0.5\%$ interval. An even 50/50 split yields a correct linkage rate of almost 85%. Yet, this split becomes biased towards a smaller value p. This can be explained by having a larger sample in the training database $\mathcal{DB}$, on which the traces of $\mathcal{DB}'$ are queried on. However, for $p = 0.1$, this accuracy drops significantly. This can be explained by the previous experiments, showing that a sample of as little as three observations suffices for a high classification accuracy. However, since many of the traces only have $10 - 20$ observations, there is a high chance that a 10% random sample may only have one or two observations.

For the Instagram-Twitter matching, the results are shown in Figure 6.7(b), for the two cases of using the data as is, thus having all Instagram observations also present in the Twitter database, and the case of splitting the dataset, thus removing the Instagram observations from the Twitter traces. Using the raw dataset a prediction accuracy of roughly 80% using $k = 1$ nearest neighbor classification to build the bi-partite graph is observed.

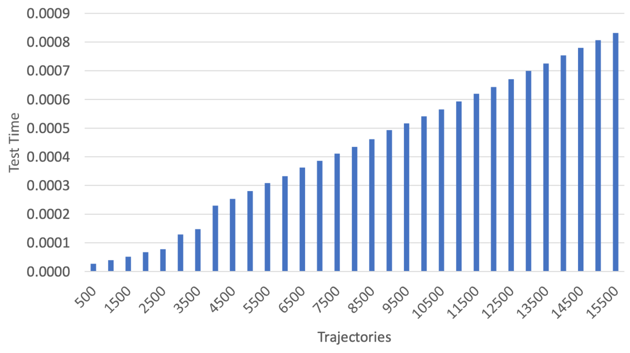
In contrast, the case of splitting Instagram off of Twitter, the accuracy drops to about 10%. These disappointing results can be explained by making the hypothesis that users use Instagram and Twitter in different ways, such as using Instagram when on a far-away vacation, while also using Twitter in locations where you don't usually take a picture, such as work and home. Also, some of the users had all their tweets linked to Instagram, such that the algorithm had no training data left in the Twitter database, thus having to random guess the user. Thus, it appears that Twitter and Instagram are used differently by users, making the Instagram sample much harder to match than a uniform random sample taken from Twitter.

Scalability

In all of the previous experiments, only the top 500 Twitter users in London were used. In the final experiment, this number of users is scaled up, by using 15,989 users that have a least two trace containing at least two observations each. This larger dataset contains over 2.7 million Tweets, including the original dataset. Statistics for this dataset are shown in Figure 6.6. The quality of the observed traces is much worse compared to the earlier 500 users explored in Figure 6.2: In Figure 6.6(a) more than half of the users have less than five traces within the twelve epochs, and only a small fraction of 6% of the users have maximum number of twelve traces. In addition,



(a) Classification Accuracy



(b) Run-time (in seconds)

Figure 6.8: Scalability: Scaling the number of Twitter users.

Figure 6.6(b) shows the quality of these trace is much lower, as nearly 50% of the traces have three or less observations. Due to the quality of this data a eight-fold split was no longer possible. A stratified shuffle split was used instead, taking 10 iterations of 20% samples.

The results on this dataset, in terms of classification accuracy as well as run-times are shown in Figure 6.8. In terms of accuracy, there is a vast decrease in accuracy observed, even for the default setting of 500 users. This is because the experiments are no longer using the top users, but just a random sample of users, and the data quality, in terms of number of observations per trace, as well as the number of trace per user, is much lower for these users.

Clearly, less frequent users are harder to classify, since there is less information. As the experiments are scaled up the number of users, there is a decrease in classification accuracy, as the classification problem becomes harder having more users. Still, the classification accuracy remains at almost 50%, despite the large number of 15,989 users, and the much lower trace quality.

Since a kNN classification is employed, and thus a lazy learning method is used, there is no model learning phase. The run-time results for the classification is shown in Figure 6.8(b).[3] a linear run-time is observed, which is attributed to the extreme high dimensionality of the feature vectors, which cannot be beneficially supported by an index structure for the kNN search. But even at the full 15,989 users, the time to classify each trace is less than $1ms$.

6.5 Conclusion

In this chapter, the challenge of identifying users in a spatio-temporal database was approached. This approach uses historic traces of a user to learn their motion in space and time, by proposing various feature extraction and similarity search methods. Using a 12-week dataset of Tweets in the London region, the experimental results show that it is possible to map a trace to a ground-truth user with extremely high accuracy.

This raises various concerns and opportunities:

- **The Threat** of loss of privacy: Traces of real people are publicly available. Given only few observations of an individual. For example, one person inadvertently appearing in the background of another person's Facebook images, then identifying this user in a trace database, and linking them to additional data, such as username or real name.

- **The Potential** through record linkage of trace databases. For example, joining the personal interests in locations (such as restaurants, bars, cafes) from a LBSN with textual thoughts of a user from a microblog. Thus, a micro-blog tweet from user u such as **"I'm visiting the Oktoberfest with my LMU colleagues!"**, might be used to recommend restaurants to u by mining their restaurant preferences using their check-ins in the LBSN.

- **The Challenge** of privacy preservation by trace obfuscation and other means. By learning the characteristics that make a trace matchable to its user, techniques to hide particularly descriptive and discriminative observations from the public trace can be developed.

[3] Run-time tests were performed on AWS using a m4.2xlarge EC2 instance running Amazon Linux. This instance type has 8 CPU cores and 32GB of RAM.

Chapter 7

Socio-Textual Mapping

The work presented in this chapter has been published as the article *Socio Textual Mapping* in the Proceedings of the 8th ACM SIGSPATIAL International Workshop on Location-Based Social Networks, 2015 [262].

7.1 Introduction

Traditionally, a spatio-temporal database consists of triples (objectID, time, location), mapping objects (e.g., users) and time to a position in geo-space where the object was, is, or will be located. In recent application, this geo-information is further enriched by textual information: For example, in geo-social networks user can check-in at their current location such as a restaurant and publish a textual description of their experience at this location. Another example is Twitter where users can broadcast small messages of no more than 140 characters. Many of these *Tweets* contain a geographical tag corresponding to the geo-spatial position of the user. Loosely speaking, the textual content of a Tweet contains information about *what's on the mind of a user*: For example, a Tweet may describe an experience that a user wants to share, a restaurant that a user wants to recommend, an achievement that the user wants to boast about, or simply anything the user wants to say. In this paper, we want to generalize this concept, by making the assumption that the collection of recent tweets of a region reflects *what's on the mind of a region.*

As an example, consider the two topics *"Justin Bieber"* and *"Greek Bankruptcy"* and consider two geo-spatial regions, such as *Ontario, Canada* and *Germany.* It may turn out that in Ontario, one percent of all tweets contain the keyword "Justin", and five percent of all tweets contain the word "Greece". In contrast, the Twitter users in Germany may user the keyword "Justin" in only

0.1 percent of their tweets, but use the keyword Greece in ten percent of their tweets. Clearly, these two distributions of keywords are different. Thus, people in Ontario and people in Germany have different things that they tweet about - different things that are on their mind. We want to automatically extract a feature representation of what's on the mind of people.

In the past, such a vision of describing a region by text messages published in that region was entirely infeasible. Even in the example above, if we only have a few hundreds of tweets per day in Germany, then making significant statement about the frequency of the topic *"Justin Bieber"* is hard. Trying to make conclusions about the frequency of rare topics such as "Databases" was hard. Drawing conclusions for smaller spatial regions, such as cities or parts of cities was completely impossible. But now, both the current trends in technology such as smart phones, general mobile devices, stationary sensors and satellites as well as a new user mentality of utilizing this technology to voluntarily share information produce a huge flood of geo-textual data. Today, we have 500 million tweets per day[1] which, in addition to other sources of geo-textual data such as travel blogs and social networks, we are suddenly able to make significant conclusions about the frequency of rare terms even in small spatial regions. It's time to use this data. In [69], Cheng et al. proposed a framework to predict a twitter users city-level based solely on words in corresponding tweets. We generalize this idea to not only determine words that classify cities, but find the latent concepts and topics that describe a generic region. Thus we extend the relationship to areas such as districts, cities, states or even artificial regions not tied to political borders (e.g. a music festival event at an off-site location). Our vision is to describe geo-spatial regions by a representation of their thoughts. We therefore derive simple representations from the textual contents of microblog data. Using this representation, we want to hierarchically cluster the world in terms of what's on the mind of their people. We call the resulting a *socio textual map*, envisioned to be useful in a large variety of application fields:

- Research in sociology has focused on the problem of Ghettos and social tensions in modern cities [130, 75, 131]. Data used in this kind of research uses Census data using "up to six race/ethnicity groups (white, black, Hispanic, Indian, Asian and other)"[131]. We claim that, especially in the 21st century, the race/ethnicity distribution of regions is *not* the sole source of social tension. Social tension may be caused simply by having different opinions and beliefs. With our solution, we can find spatial regions, on a city scale, having people with significant different interests. This may or may not be a result of ethnic differ-

[1]`https://about.twitter.com/company`

ences. Our proposed approach contains much more facets of people, by directly mining the interests of the crowd.

- Our research may improve the process of geocoding of geo-textual data. Given a user who specified "London" as his location, the probabilistic distribution might be shifted towards the city of London, Ontario, Canada, if the vocabulary, topics and keywords of his tweets are more similar to regions within that area. This can be done by describing the user, who is to be geocoded, by the set of his own tweets, obtain a proper feature representation and compare this representation to candidate geo-locations.
- For targeted marketing, it may be much more interesting for a company to direct their advertisements to an area of people having a similar mindset. Even if this region covers multiple political regions. For example, an upper-class car manufacturer may be looking to direct an advertizement at a wealthy city district. However, parts of the administrative city districts may not actually wealthy, or the actual wealthy population may reach outside of the city district. With our approach, the car manufacturer can target it's advertizement at the mental cluster that is rooted in the wealthy city district.

Obviously, the performance of the hierarchical clustering step strongly depends on the quality of the text representation vectors that are used to fit the clustering model. Although we limit the proof of concept to fairly simple representations, summarized in Section 3.4, we want to emphasize that more sophisticated text mining approaches as for instance those surveyed in [18], or even deep learning approaches that learn vector representations [200] might boost the performance of the clustering step.

The remainder of this chapter is as follows. In Section 7.2 we formalize our search for a socio textual map, and identify the research challenges that need to be solved towards this vision. In Section 7.3, we implement a first solution, by solving each of the research challenges in an initial way. We show that our vision is feasible: if the necessary research steps are all solved thoroughly, then a large scale solution to map the minds of people is a vision that may become reality.

7.2 Socio Textual Maps

In this section, we formally define our notion of our vision of a socio textual map. We present a theoretical foundation to prove why the concept of a

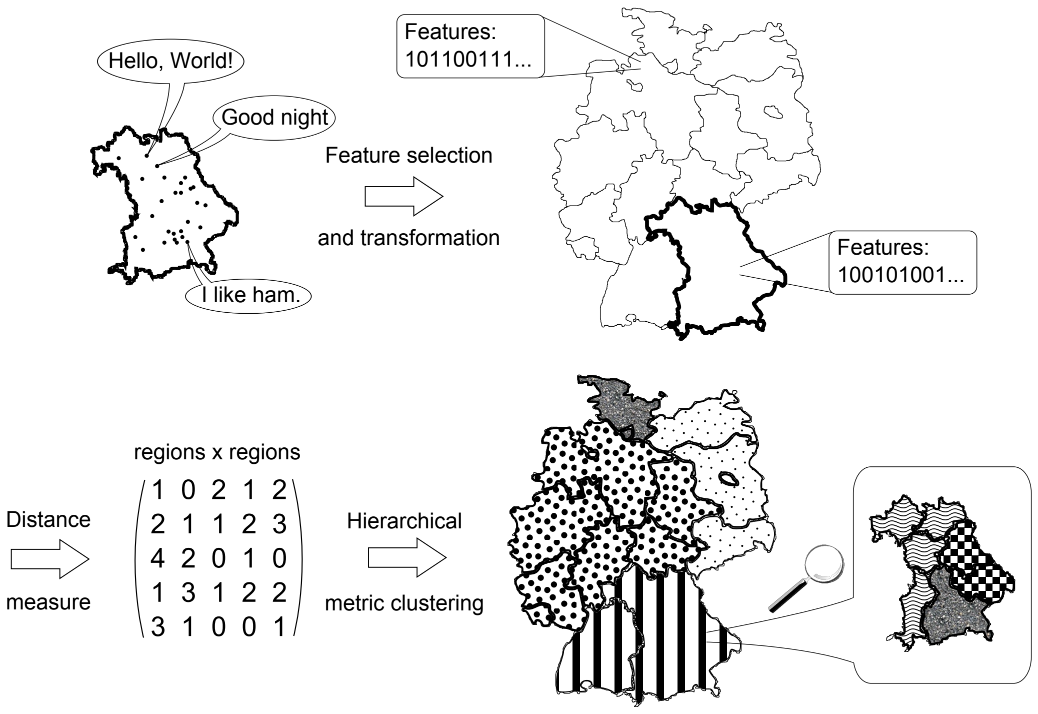


Figure 7.1: Searching in collections of multi-represented users.

socio textual map is feasible and discuss problems, open research questions and challenges. The first step requires to obtain a feature representation of a potentially large and dynamic set of textual documents.

Definition 20 (Feature Selection)**.** *Let $\mathcal{S} \in String^*$ be a set of text documents. A function $f : \mathcal{S} \mapsto R^d$ is called a d-dimensional feature representation of $\mathcal{S}$.*

The choice of function f is one of the main challenges. This function should chosen such that two of text documents $\mathcal{S}_1$ and $\mathcal{S}_2$ are similar in terms of the topics, interests and experience of these texts, if and only if $f(\mathcal{S}_1)$ is similar to $\mathcal{S}_2$. Thus, a proper feature selection method should discard terms without informative content, i.e. words that appear very frequently. A common approach is referred to as document frequency-based selection, introduced by [175]. The idea is to weight those terms that appear more frequently with a higher value. To avoid prioritizing stop words like *the*, [238] extended this approach by using *inverse document frequencies*. However, these approaches give no information about the importance of a keyword in terms of describing the mental topic of the user generating the text. This is the challenge of feature extraction for socio textual mapping. In [76] a concept is presented that uses an entropy measure to select those features that carry the most information. Taking this concept to feature selection for geo-textual data, the idea is to select terms which are highly frequent in only a few regions, and extremely rare in others. Such local trends may be extremely useful to distinguish regions at a local scale, but may become useless for other scales and other areas. However, it seems intuitive that a proper approach should also include global trends, that most of the world has (to different degrees) on their mind.

Next, we apply function f to geo-spatial regions.

Definition 21 (Feature Transformation). *Let $\mathcal{W}$ denote a hierarchical partitioning of the geo-spatial region representing the surface of the Earth. Each level of $\mathcal{W}$ corresponds to a geographic scale, i.e., continental level, country level, state level and city level. For each region $w \in \mathcal{W}$, the function $text(w) : \mathcal{W} \mapsto String^*$ returns a set of text documents that are associated with w.*

The first step of our workflow in Figure 7.1 illustrates this step. In the top-left of Figure 7.1, we consider a spatial region w corresponding to Bavaria, Germany. We take the set of text messages $text$(Bavaria) and apply a (in this example binary) feature transformation. The same feature transformation is performed to all other German states. In the next step, we need to assess the pair-wise dissimilarity between these regions, using standard vector distance functions. Using the resulting dissimilarity matrix, exemplarily depicted in the lower-left of Figure 7.1, we can apply a metric clustering approach to find groups of similar regions. The choice and the parameterization of this clustering approach are another challenging step. For instance, the clustering approach needs to account for different geographic scale. That is, regions on country level should allow much more freedom to be considered similar than regions on a city level.

Theoretic Foundation

There are two main theoretical reasons why finding a socio textual map is viable and feasible. The first is *the law of large numbers*, and the second is Tobler's first law of geography.

The law of large numbers states that, for a random variable, the empirical probability approaches the actual probability as more trials are performed. Applied to our problem, we can treat the topic of a tweet as a random variable. For a sufficiently large number of tweets drawn from a region, the law of large number states that the fraction of tweets having a specific topic con-

verges to the true fraction of people having this topic on their mind.[2] The flood of daily tweets and other geo-textual data sets allows us to exploit the law of large number to obtain a representative sample of the minds of people in a region.
Tobler's first law of geography states that "Everything is related to everything else, but near things are more related than distant things" [248] and is one of the key reasons why "spatial is special" [162]. It is the reason why we expect that a clustering of the minds of regions results in a clustering that is also spatially correlated. And it is the reason why we envision that we can obtain a socio textual map that captures more than lingual vocabulary, but also captures topics and trends that people think about.

7.3 Proof of Concept

For a proof of concept we collected tweets with geographic metadata from Twitters public streaming API[3]. On each tweet's text we applied the Apache Lucene standard tokenizer to extract word tokens. For each geo-coordinate we use the reverse geocoding library[4] (based on OpenStreetMap data) to obtain the corresponding city. In Section 7.3 we present an initial solution to derive features corresponding to the relevance of terms based on their relevance to a city. In Section 7.3 we will present different clustering results which show that nearby cities are generally more related than cities further away from each other.

Feature Selection and Transformation

Each tweet t is represented as a tuple $(c_t, W_t) \in S$ where c_t denotes the city corresponding to the tweet's location and $W_t = \{w_{t,1}, ..., w_{t,n}\}$ denotes the set of extracted word tokens for the text of tweet t. After each epoch (say, 1 hour), we want to obtain a set of k most "representative" features for each city c. Using only the top-k most frequent terms would result in a sole

[2]Two implicit assumptions are made here. First, we assume that a the content of a tweet correlates with what on the mind of the tweeter; second, we assume that sample tweets are drawn independent without any bias. The second assumption is critical, as some people are "more vocal" than other, thus tweeting more often than others and thus, overrepresenting the topics on their minds. However, it was shown in [241] that the law of large numbers still holds as long as there is no user which *dominates* all other users in terms of tweet frequency, and as long as the error is unbiased, i.e., the frequency of tweets of a user is independent to the topics on his mind.

[3]https://dev.twitter.com/streaming

[4]https://github.com/kno10/reversegeocode

clustering of languages, rather than a clustering of people's thoughts, because regions are usually dominated by the English language and thus, stopwords like the, and, or, etc. would cause regions of actually different mind sets to be indistinguishable among themselves. Let tf_w and $\mathrm{tf}_{w,c}$ be the frequency of word token w and the frequency of w mentions within a city c. We then define a score of a word token w as follows:

$$\mathrm{score}(w) = \frac{P(w|c) - P(w)}{\sqrt{\mathrm{tf}_w}} \mathrm{tf}_{w,c}$$

where $P(w)$ denotes the probability of obtaining the word token w regardless of its location and $P(w|c)$ the probability of obtaining word token w given a city c respectively.

Clustering

To respect the hierarchical nature of our data we utilize an agglomerative clustering approach. Therefore we first construct a similarity matrix of the size $n \times n$, where n is the number of all cities. The cell $c_{i,j}$ for the i-th and j-th city is set to the Jaccard similarity coefficient of the corresponding feature sets F_i and F_j, i.e. $c_{i,j} = \frac{|F_i \cap F_j|}{|F_i \cup F_j|}$. As shown in Figure 7.2(a) we are able to determine political country boundaries. Each circle represents a city having at least 150 tweets in a time-frame of 60 minutes. On a finer geographical scale, we obtain more detailed micro-clusters within countries in Europe, such as Great Britain in Figure 7.2(c)), and France and Spain in Figure 7.2(d). On a even more detailed scale, we can find clusters within a city, such as the city of London (England) in Figure 7.2(b). These different clusterings are obtained by varying the distance threshold parameter of the clustering algorithm. It is important to note that the selected features do *not* consider spatial distances. Nevertheless, the clusters that we obtain, in all settings of Figure 7.2 are, more or less, grouped in geo-space. We contribute this result to Tobler's first law of geography as presumed in Section 7.2.

The main conclusion drawn from this evaluation is that it is possible to invert Tobler's law: We show the counter-direction that regions having similar tweets are also more like co-located in geo-space. For instance, in the experiment of Figure 7.2(e) we only consider tweets that are flagged to be of Spanish language. We see that cities in Spain form a cluster that, except for a few outliers, represents all of Spain and Spain only - and cities in Mexico form a cluster that, for the most part, represents all of Mexico and Mexico only. Note that this experiment also shows some cities in the United States, as these may also have more than 150 tweets in Spanish language over the considered time period. This observation is a proof of concept for the vision of socio textual mapping using user data to describe the mind of people of a region.

7.4 Challenges

Now that the concept is defined, theoretically founded and an empirical proof of concept is provided, we identify some of the challenges that need to be addressed by the research community for this vision to become a success.

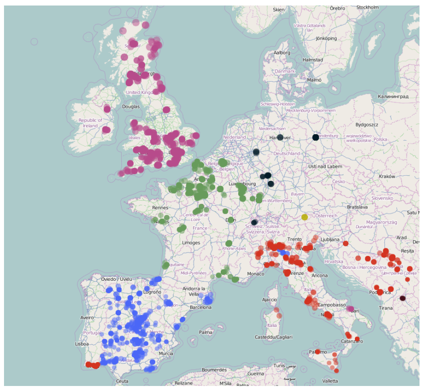
Feature Representation: We need to find a feature representation of tweets of a region that capture the mind of the people of that region, rather than their vocabulary. In our proof of concept, we used the most frequently used terms in a region. Clearly, any region in the US or any English speaking country will frequently use *stop words* such as “the” and “and”. While our proof of concept in Section 7.3 removes these stop words, it is still an open question weather the remaining keywords are representative for the interests and thoughts of users. A more desirable approach, which will allow to better reflect the mentality of a user, rather than his vocabulary, one could look into solutions for temporal-textual trend mining as presented in [57, 227]. This way we could directly count the mentions of topics that are, globally, on the mind of people.

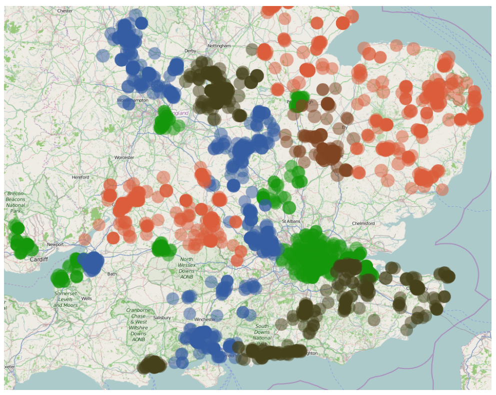
Distance Measure: Given occurrences of appropriate set of keywords, or any other means of representing the mind of region, we need adequate solutions to measure the similarity between regions. In our experiments, we simply used a Jaccard index to measure similarity between the selected features sets of a region (as discussed in Section 7.3). We need a distance measure which also takes into account the geographic scale of the considered regions: Two small parts of a city should penalize minor difference much more drastically. Also, a good distance measure may also consider the spatial distance of the regions, thus incorporating Tobler’s first law of geography in the distance function.

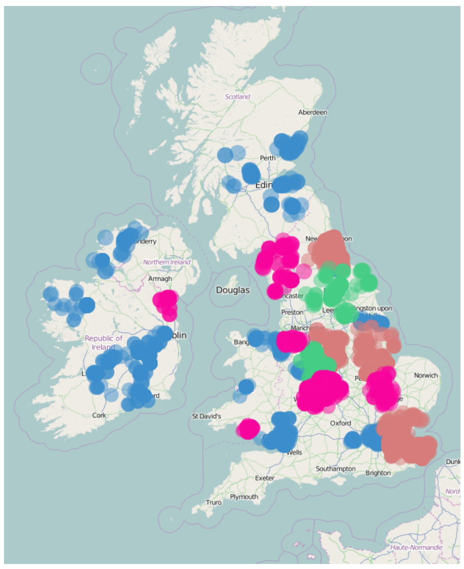
Metric Clustering: We use an agglomerative single-link clustering, i.e., a single-link clustering that returns a Dendogram of different clusterings for different single-link-distance parameter values. This result allowed us to manually pick single-link distance thresholds for different geographic scale. For example, we manually picked a proper single-link-distance value to obtain a good clustering on country-level, and a different value for a good clustering on city-level. While our proof of concept showed that this proper parameter choice led to solid results which were highly correlated to political (and non-necessarily lingual) borders. Yet, an approach is needed to automatically adapt its parameters for different scale-levels. Also, it would be great to compare different levels, such as large cities and small countries.

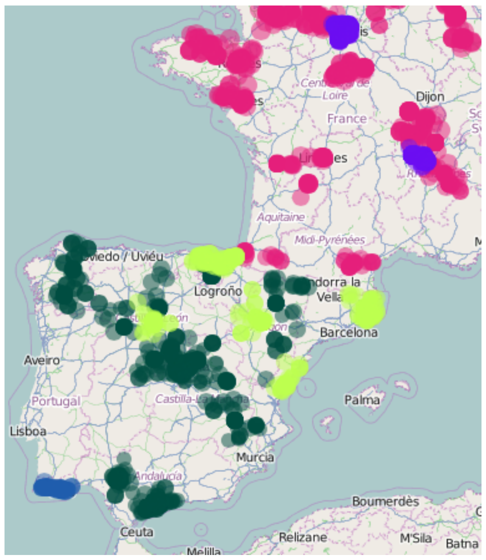
Other Data Types: For our socio textual maps, we exclusively used geo-textual data to describe the mentality of a spatial region. We made this choice since large sets of geo-tagged and user-tagged are available publicly. But this is not the end of the vision. Other data types can be used to estimate the mindset of a region. For instance, time-series of activity of users, content of published multi-media data, attributes of users of a region (e.g. age, nationality, etc.). We envision a feature representation to capture all social information available, in order to plug this feature representation into the framework of Figure 7.1.

Independence of languages: We see that lingual borders also imply mental borders. Ultimately, we envision a solution that is completely independent of languages. Thus, we want a region in Japan to be able to cluster with a region in Spain, if and only if both regions have the same topics in mind, but use different vocabulary (and letters) to describe the thoughts on their mind. Thus, we envision feature representation that are not merely based on vocabulary, but more precisely capture the mentality of people. We strongly believe, that a joint effort of the GIS community will allow to overcome these challenges to manifest our abstract vision of a socio textual map into a powerful instrument for social good.


(a) Clustering at a high level yields country boundaries.


(b) Language independent micro-cluster at city level (e.g. London).


(c) Language independent cluster results at country level.


(d) Clusters within Spain and France

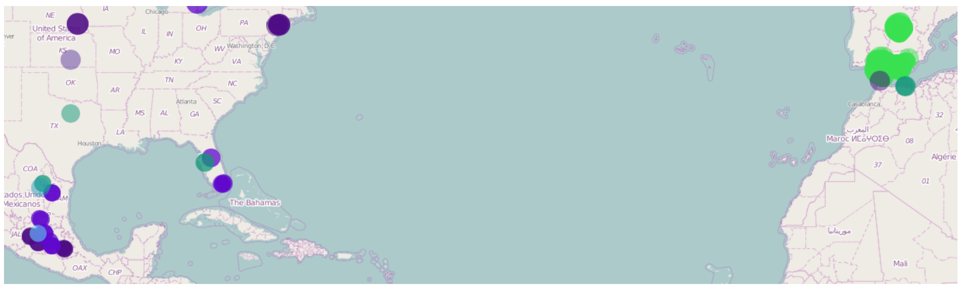

(e) Separation for same Language (Spanish) into distinct clusters.

Figure 7.2: Visual clustering results.

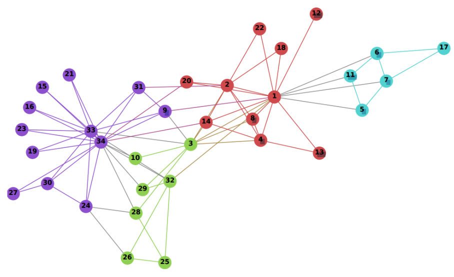
Part II

Representation Learning on Graphs

Chapter 8

Introduction

One data structure that we want to put special emphasis on in this part of the thesis are graph structures, respectively algorithms that process data that are modeled as graph structures. In general, graphs are especially useful when dealing with relational data as they are designed for the purpose of modeling relations between entities explicitly. In terms of graph terminology, such entities are usually referred to as *nodes* or *vertices*, while relations or interactions between entities are modeled as *links* or *edges*. Representing data as graphs provides a handy, and oftentimes efficiently accessible way to store and process data in a huge variety of application fields like biology (e.g. drug design), marketing (e.g. recommendation systems), or social sciences (e.g. social network analysis). Reasoned by their broad availability and the huge amount of information that can be captured by graphs, a very important, and also highly demanded key challenge is to represent graphs or parts of a graph, e.g., subgraphs, nodes, or edges, as numerical feature vectors such that they can be used for downstream machine learning algorithms. However, representing the wealth of information that is given by the graph structure within appropriate feature vectors is not straightforward. Traditional methods therefore often rely on hand-crafted features, like the clustering coefficient or in-betweeness scores, that quantify specific structural properties. A major drawback of these methods is that the choice of the right features is not obvious and also, that some of these structural features tend to be computationally expensive to calculate, especially for large networks. To overcome those shortcomings, a recently popular line of research focuses on *representation learning on graphs*. The key idea of all approaches that fall into this area is that structural features shall be learned instead of computing them explicitly. The different learning procedures are typically unsupervised, can mostly be framed within classical encoder-decoder frameworks [109] and


(a) Input: Karate Graph

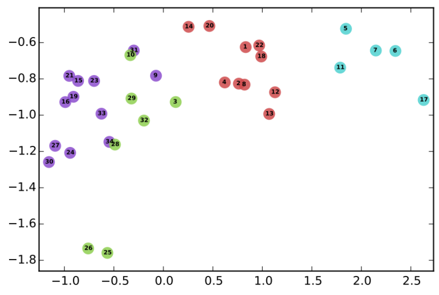
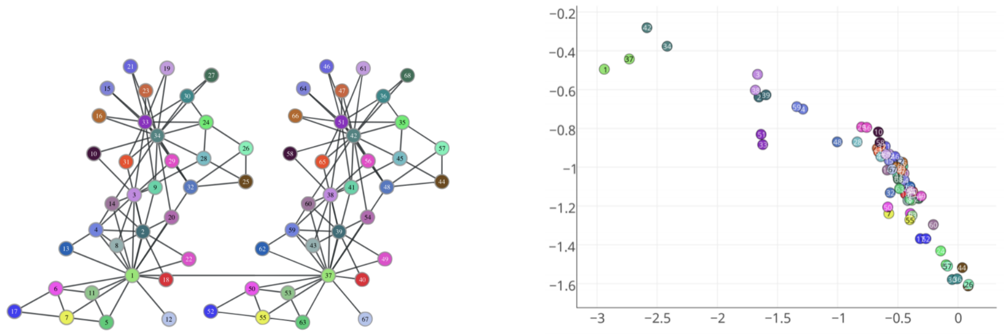

(b) Output: Vector Representations

Figure 8.1: Example for homophily-based node embeddings.

partially rely on various inductive biases[1], hence aiming at encoding different structural properties within low dimensional feature vectors in $\mathbb{R}^d$. In general, the different inductive biases guide the learning processes so that the trained models optimize the learned mappings such that specific structural properties that are present in the graph are also reflected in the learned vector space.

Focusing on the task of learning representations for nodes in graphs, the general goal is to learn functions that map vertices in the graph to low-dimensional vector representation, i.e., $V \rightarrow \mathbb{R}^d$, with V denoting the set of nodes in the graph. The inductive biases can be divers. One common inductive bias is the so-called homophily assumption. Encoder models that follow this assumption are biased towards learning node embeddings such that nodes which reside in the same communities within in the graph are also similar (close to each other) in the vector space, while the representations of nodes from different communities are dissimilar. Figure 8.1, taken from [214], depicts an outcome of such an homophily-based node embedding for the Zachary's Karate Club network, a social network where nodes denote the members of a karate club and links indicate social interactions between the members. In the given example, the nodes have been mapped into a 2-dimensional latent representation space. One can observe that the community structure is preserved within the vector space, i.e., the node representations of the nodes that belong to the same community may end up in the same spatial cluster while nodes from different communities may belong to different spatial clusters. Once having this kind of node embeddings after the encoding step they are typically fed into the decoder to make specific

[1]Inductive biases can generally be understood as assumptions that are given into the learning process, e.g., by the model architecture, the loss functions or other components, as priors (see [28] for details).



(a) Input: Mirrored Karate Graph (b) Output: Vector Representations

Figure 8.2: Example for structure-based node embeddings.

predictions. Classical prediction tasks include node classification and link prediction. For the latter task, the common approach is to combine pairs of node embeddings into a single edge representation and subsequently use these representations for a binary classification. The basic assumption behind using homophily-based node representations for prediction tasks is that class distributions correspond to the community structure within the graph. This assumptions holds for many applications. For instance, in a social network that models blogging platforms where nodes indicate users, edges link users that discussed a specific topic and the labels or classes indicate interest groups, one may rely on homophily-based node embeddings to predict the interests of an user, or to make recommendations that point users to specific topics.

Another well-known inductive bias for node representations that we focus on in this part is the topology-preserving bias. Node embedding techniques of this class aim at encoding local topological properties so that nodes that have a similar topology within their local neighborhood end up close together in the latent vector space. The idea of this type of node embedding is visualized in Figure 8.2 that is taken from [217]. On the left side of the figure is the mirrored Karate network where the original network is simply copied and both copies are connected via a single edge that links one node of the network to its exact copy. Each node and its corresponding copy have the same color, and in particular each node and its corresponding copy also have the same topological structure within their local neighborhoods. The right figure again shows the 2-dimensional latent vector space, and it can be observed that nodes having the same color are mostly spatially close to each other.

In contrast to homophily-based approaches, encoding nodes based on their local neighborhood topology is less useful when aiming at solving a task as described above. However, imagining the task of recommending “follow” relationships in a social network where nodes are users and “follow” relationships are modeled as undirected links, structure-based encodings may indeed be more useful than homophily-based embeddings. This is due to the inherent multi-modal[2] characteristics in such social networks. Considering the Twitter network for instance, all users can simply be seen as users. However, it can be observed that some users are different than the majority of users in terms of structural properties within their local neighborhood. E.g., the nodes that correspond to celebrity-users typically have a much higher degree than “usual” fan-users. Naturally, it occurs that such celebrity-users and their fan-users form some sort of community. So, using homophily-based embeddings for predicting new “follow” relationships for a given fan-user in this scenario would lead to link recommendations to other fan-users with a high probability. This may be useful if the task is to connect a specific community, but less useful if the objective is to make some sort of celebrity recommendations like “if you like rockstar A you may also like rockstar B”. In such cases, structure-based node embeddings are more appropriate. A somewhat more complicated use-case for structural node embeddings is the classification of proteins according to the function they have within protein-protein interaction networks.

The remainder of the second part of the thesis presents algorithms that have been developed in the area of representation learning on graphs during this thesis' work. After discussing related work in the field of representation learning on graphs in the following chapter, we present an unsupervised approach for learning homophily-based node embeddings based on approximated Personalized PageRank distributions in Chapter 10. After this, we present an unsupervised method for calculating structure-based node embeddings based on multiple local neighborhoods of various extents in Chapter 11. In Chapter 12, we present a clustering-based approach that uses the structure-based node embeddings from the previous chapter to embed entire graph structures. Chapter 13 shows our work on semi-supervised node classification that aims at predicting node labels from partially observed labels within multiple local neighborhoods. Finally, to conclude this part we present results of our study where learned node embeddings have been applied for the task of map conflation in Chapter 14.

[2]Multi-modal in the sense that there actually exist different types of nodes without the nodes being tagged so explicitly.

Chapter 9

Related Work

Due to the broad range of applications for relational data represented as network structures, there has also been a large body of work on solving embedding problems. In this chapter, we briefly survey previously published works that are related to the topics that are discussed throughout this part. In detail, we first discuss related work on unsupervised node embedding, before we focus on methods that have been developed for learning node representations in a semi-supervised manner. Finally, we also review published work that aims at embedding entire graphs for graph classification.[1]

9.1 Unsupervised Node Embedding

The pioneering approaches in the field of unsupervised node embeddings mostly use global, spectral methods with the purpose of finding structure-preserving, low-dimensional embeddings for vertices in a graph. Notably, these approaches originally aim at solving the task of (non-linear) dimensionality reduction in non-relational data. Therefore, the general inductive bias of these methods is that the data lies on a low-dimensional manifold. However, they are related to node embedding techniques as they firstly construct some sort of similarity graph, and subsequently feed either the node similarity matrix or the graph Laplacian into some matrix factorization step. In a very general sense, the classical *Multidimensional scaling (MDS)* algorithm [249, 102] can be understood as such a methodology, since the idea is

[1]As this chapter mainly discusses works that are closely related to the methods presented in the upcoming chapters, and in particular does not cover works on e.g. heterogeneous graphs, or edge embedding methods, or supervised methods (e.g. [224, 268, 159, 165, 247, 142]) that also fall into the field of representation learning on graphs, we refer the interested reader to a recent survey that can be found in [59].

to factorize the (dis)similarity matrix of objects such that similar objects are represented close to each other in the resulting low-dimensional space and dissimilar objects are far from each other. In fact, the similarity matrix can be seen as a complete network, where each pair of nodes (i.e., objects) is connected. Following their intuition, multiple methods, e.g., *Isomap* [246, 27], *LLE* [220], *Laplacian eigenmaps* [30], or *LPP* [116], refine this idea with the intention to preserve structural properties of the underlying data. By constructing a kNN graph, where each data object is related to their k-nearest neighbors in the feature space (with respect to some feature-based similarity measure), and deriving a similarity matrix that is then factorized, they finally get low-dimensional representations. These approaches mainly differ in how they derive the similarity matrix, however, all aiming at capturing local, structural properties.

More related to the works presented in this part of the thesis are the recently proposed node embedding techniques that mostly rely on neural learning and have explicitly been designed for embedding graph vertices. Here, we broadly categorize those works into homophily-based node embedding and structure-based node embedding techniques as this reflects the structure of the following chapters. However, please note that there might be methods that allow for the flexibility to be biased towards either of these two categories.

Homophily-Based Node Embedding

In general, all homophily-based node embedding techniques aim at preserving community structure, and hence they mainly rely on the local structure, respectively the local node neighborhoods. Therefore, they mainly differ in how to determine the node neighborhoods. The first method that has been developed for learning node embeddings based on the homophily assumption is *DeepWalk* [214]. The key goal of this method is to learn latent vector representations for nodes based on the local neighborhoods (in the network domain) of the nodes. Precisely, the algorithm first performs a couple of random walks that are intended to capture the structural properties of the underlying graph. After performing the random walks, the authors propose to slide a small window over the retrieved node sequences and sample "focus-context" node pairs from the window. The intuition is that the "context node" is assumed to be a node appearing in the local neighborhood of the "focus node". Finally, the pairs are used as training instances to learn the homophilic node embeddings by using the so-called *SkipGram* model [182, 181] (we postpone the discussion of this model to Chapter 10). This way, *DeepWalk* is rather likely producing similar embeddings for nodes with similar neighborhoods. Based on the *DeepWalk* framework, Grover et al. presented the *node2vec* algorithm [103]. The key difference lies in the way *node2vec* explores the local node neighborhoods. Instead of relying on classic random walks, the authors propose to use second-order random walks with the goal to increase the adaptability to differently structured neighborhoods. This flexibility comes to the cost of introducing two hyperparameters that guide the random walk. By setting these parameters, users can bias the walks towards a breadth-first search or a depth-first search strategy, which in turn introduces different inductive biases. While a breadth-first search strategy tends to lead to homophily-based embeddings, a depth-first search strategy tends to lead to structural embeddings. However, we want to note that in most applications a proper hyperparameter setting is not straightforward and hence costly grid-searches (that possibly require ground-truth data) are usually necessary. The *LINE* algorithm [244] learns two different representations. The first-order proximity representation is learned by maximizing the joint probability of directly connected nodes. Therefore, the directly connected nodes have to be close to each other in the vector space. As stated by the authors, this learning goal is only suitable for undirected graphs. For the second-order proximity the conditional probability of the node given its direct neighbor is maximized. This results in embeddings, where nodes are close to each other in the vector space if they share the same direct neighbors. Thus, the second-order proximity representation is similar to the *DeepWalk* result when considering only the direct neighbors for each node. To incorporate higher-order proximities, too, the *GraRep* algorithm [62] computes a sequence of matrices, i.e., random walk transition matrices taken to powers ranging from 1 to k, and subsequently applies SVD to them. In contrast to the previous homophily-node embedding methods, *GraRep* uses explicit instead of implicit factorization of co-occurrence matrices. The recently proposed *VERSE* algorithm [253] uses a generic but fixed node similarity measure specified by the user as input to learn the distribution of the given similarity measure that, in turn, is used to generate the embeddings. In particular, the fact that the user can specify the similarity measure means that *VERSE* also allows for the flexibility to range between homophily-based and structural node embedding. Further recent methods focus on additional aspects, notably the authors of *GraphSAGE* [108] propose an inductive node embedding technique that relies on additional node attributes. *Graph2Gauss* [43] aims at taking uncertainty into account by embedding nodes as Gaussian distributions.

Structural Node Embedding

In contrast to homophily-based node embedding, the idea of structural node embedding is to relate nodes if their connectivity patterns to their respective neighbors exhibit similar structural properties. To this end, *struc2vec* [217] trains a *SkipGram* model using degree sequences in neighborhoods of increasing size. However, the method hardly scales in terms of graph size. Similarly, *DRNE* [254] sorts neighboring nodes by their degree and feeds the resulting sequences into an LSTM. The authors show that in some special cases, the resulting embeddings satisfy regular equivalence which recursively defines two nodes in a graph to be role-equivalent if their neighbors have the same roles. While such graph-based role definitions are rather strict and often do not apply in the real world, we take a more flexible feature-based embedding approach. *RolX* [117] is a feature-based approach which relies on handcrafted structural features (such as node degree or clustering coefficient) and computes soft-assignments of nodes to a predefined number of roles using matrix factorization. For a more in-depth discussion on role discovery, we refer to [219]. A diffusion-based approach is taken by *GraphWave* [82], where the graph is first transformed to the spectral domain in which the signal is then filtered with a heat kernel. Approximation using Chebyshev polynomials of order up to k results in linear time complexity and a k-localized filter. Node embeddings are derived by aggregating the rows of the resulting wavelet coefficient matrix while controlling the spread of the diffusion implied by the heat kernel. For detecting patterns at multiple scales, embeddings resulting from different scaling parameters are computed and concatenated.

9.2 Semi-Supervised Node Embedding

In applications where some of the labels are known, unsupervised node embeddings can generally be used within a semi-supervised learning pipeline consisting of separate optimization steps as is the case for semi-supervised learning with pre-trained node embeddings. However, learning node embeddings via end-to-end training usually leads to better performance on the downstream supervised learning objective. To this end, traditional methods typically rely on some sort of propagation mechanism to spread label information throughout the network. *Label Diffusion* [284, 285, 281, 208, 259] methods propagate labels from labeled nodes until convergence and classify based on majority vote. While classical methods [284, 285, 281] rely on the homophily assumption, *Cosine Label Propagation* [208] applies label propagation to a similarity graph constructed from the cosine-similarity of nodes' adjacency vectors. The underlying assumption behind this approach is that nodes should have similar labels if they have similar neighbors. The authors further propose the *Two-Step Label Propagation* method [208] which skips immediate neighbors in the label propagation process to reflect the assumption that nodes have the same labels if they are structurally equivalent. *Dynamic Label Propagation* [259] extends the idea of the traditional label propagation objective by inducing label correlation between nodes into the transition matrix to reinforce propagation between nodes with similar label distributions. These label distributions consider a node's own labels and not labels occurring within the node's neighborhood. In particular, those advanced label propagation methods have the advantage of being able to deal with other kinds of inductive biases than only homophily. For instance, they also can model heterophily, i.e., nodes share the same label if they exhibit a single specific correlation between labels within their local neighborhoods, or even mixed patterns, i.e., nodes share the same label if they exhibit one of possibly multiple, specific correlations between labels within their local neighborhoods. Similar to the previously mentioned approaches, *Belief Propagation* [206, 144, 97] methods propagate labels but require an explicitly parametrized transition matrix from the user to model either homophily, heterophily or mixed patterns. It is noteworthy that similarly to random-walk based node embeddings, all of the above methods are not able to make use of information in distant or disconnected parts of a graph. Furthermore, graph diffusion methods do not have separate training and inference steps and hence requiring these methods to be executed again whenever the underlying graph changes. To overcome these shortcomings, the *ICA* and *GS* algorithms [229] iteratively classify unlabeled nodes with a local classifier and use nodes labeled in the previous iteration as ground truth. The more recent model *Planetoid* [271] combines the prediction loss with node embeddings by training a joint model that predicts class labels as well as graph context for a given node. It samples context nodes from the local neighborhood, similar to *DeepWalk*, as well as possibly distant nodes with a shared label.

A slightly different but important direction is semi-supervised learning on attributed graphs. The vast majority of recently proposed neural network based models can be described within a *Message Passing Neural Network (MPNN) framework* [98]. An important special case are *Graph Convolution Networks (GCN)* [56, 180, 80, 142, 25, 186, 156, 236, 256, 161] which aggregate node attributes over local neighborhoods with spatially localized filters, similar to classical convolutional networks on image data. Note that despite these methods typically taking benefit from additional information provided by node attributes, they mostly can be adapted to non-attributed versions straightforwardly and hence are considered as related. However, these methods are not able to learn direct correlations between a node's label and labels of neighboring nodes, be it homophily or a general type of correlation. Nonetheless, one might attempt to use MPNN in combination with class labels instead of attributes. This would require to ensure that a node will not at some point receive a message containing information about its own class label, which will happen after a sufficient number of message passing iterations if a node is reachable from itself (as is given in cyclic or undirected graphs). This problem is non-trivial since the MPNN framework requires the whole feature matrix for the forward pass and computes updates based on all labeled nodes in the training set. A naive approach of masking the nodes' own labels would result in a different feature matrix for each node and therefore leads to infeasible training costs.

9.3 Embedding Entire Graphs

Methods for embedding entire graphs with the purpose of graph classification can be sorted into *kernel-based* and *feature-based* methods. Kernel-based methods focus on deriving similarity models which typically perform an implicit feature transformation and rely on kernelized classification models. Feature-based models compute continuous graph representations which are fed to a generic classification model.

Kernel-based Methods

Established graph kernels include the *Random-Walk (RW)* [45], *Shortest-Path (SP)* [44], *Weisfeiler-Lehman (WL)* [231] and *Graphlet Kernel (GK)* [232]. The first two methods rely on additional node labels and count visits to nodes with a certain label in random walks or via shortest paths, respectively. The WL kernel is based on the Weisfeiler-Lehman graph isomorphism test and performs a relabeling of initial node labels. The graphlet kernel counts occurrences of small induced non-isomorphic subgraphs and does not consider additional node labels. A problem with all of the above kernels is that correlations between feature dimensions are not taken into account. This problem is addressed, e.g., in [269, 192, 11] by learning hidden representations of the substructures counted by the respective graph kernels. However, it should be noted that all of the above methods suffer from rather high complexity. Other works focus on directly specifying graph metric spaces. A classical example is the *Graph Edit Distance* [223] which minimizes the number of edit operations that are needed to transform one graph into another one. Such approaches are usually NP-hard and rely on heuristics [31, 196]. For instance,

[31] proposes a family of tractable graph metrics that approximate two common intractable metrics and provide extensions to incorporate additional node attributes. In [196], graphs are represented as bags of node embedding vectors and compared by using the Earth Mover's Distance (EMD), where additional node labels may be incorporated by matching only nodes with the same label.

Feature-based methods

Earlier works on feature-based graph classification rely on handcrafted feature embeddings. Notably, *NetSimile* [34] extracts structural features (such as node degree and clustering coefficient) for each node and aggregates them per graph using different aggregation functions including mean and standard deviation. Similar to graphlet kernels, another line of research focuses on describing graphs by decomposing them into subgraphs. *Subgraph2vec* [192] computes subgraph embeddings using a SkipGram model where subgraphs are rooted and context subgraphs are rooted at neighbors of the source root. The more recent method *GE-FSG* [194] represents graphs as bags of frequent subgraphs and learns graph embeddings using a document embedding technique. *GAM* [150] addresses the problems of scalability and noise by using an attention model to focus on small and informative parts of a graph. However, the method relies on additional node attributes. *Sub2vec* [11] also focuses on embedding subgraphs but does not attempt to represent graphs by their subgraphs.

While *GraphWave* [82] considers only node embedding, other diffusion-based methods focus on embedding whole graphs. *DeepGraph* [158] computes graph embeddings based on heat kernel signatures. However, the final embeddings are trained end-to-end for predicting network growth. In addition to the heat kernel signature, *NetLSD* [252] further considers the wave kernel signature. Similarly as in *GraphWave*, signatures of different scales are concatenated in order to obtain a multi-scale representation and a k-th order approximation is performed to make the eigendecomposition of the Laplacian scalable. Message Passing Networks [98] is a class of neural networks models for graphs. The primary focus of these methods lies on learning embeddings from node attributes and they cannot be applied out-of-the-box to classify general non-attributed graphs. *Patchy-san* [195] addresses this problem by considering auxiliary node labels such as degree or PageRank centrality.

Chapter 10

Homophily-Based Node Embedding

The work presented in this chapter has been published as the article *LASAGNE: Locality and Structure Aware Graph Node Embeddings* in the Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2018 [92]. The work has been awarded with the Best Student Paper Award. A preliminary version which serves as a technical report has been published on arXiv as preprint arXiv:1710.06520, 2017 [91].

10.1 Introduction

Graphs are a common way to describe interactions between entities. The entities are modeled as nodes, and the interactions between pairs of entities are represented by edges between nodes. Describing nodes of a graph as low dimensional vectors has the advantage that many popular machine learning algorithms can be automatically applied, and it is applicable in many areas like visualization, link prediction, classification, etc. [176, 166, 13, 38]. Motivated by this, so-called representation learning methods for graph vertices, e.g., [214, 244, 103], focus on learning vectors to represent information in neighborhoods around a node, e.g., nodes within a short geodesic distance or nodes encountered in random walks starting at a given node.

Somewhat more formally, let $G = (V, E)$ be a graph, with $V = \{v_1, \ldots v_N\}$ being the set of nodes and $E = \{e \mid e \in V \times V\}$ being the set of (undirected) edges. The general goal is to find a vector embedding or latent representation for each node v_i such that the resulting set of embedded nodes $\mathcal{E} = \{f(v_i) | v_i \in V\}$ in the d-dimensional vector space $\mathbb{R}^d$ still reflects structural properties of G. For instance, such structural properties could be the similarity of the neighborhoods of two nodes v_i and v_j. The neighborhood $\mathcal{N}(v)$ of a node v is defined as the set of nodes having the highest probabilities to be visited by a random walk starting from node v, a geodesic walk starting from v, or some other related process. This means if $\mathcal{N}(v_i) \approx \mathcal{N}(v_j)$ holds in the original graph space, it should also hold that $f(v_i) \approx f(v_j)$ in $\mathbb{R}^d$.

The intuition behind these representation learning methods is that nodes having similar neighborhoods are similar to each other, and thus one can use information in the neighbors of a node to make predictions for a given node. Defining the right neighborhood for each node, however, is a challenging task. For example, in unsupervised multi-label classification, the labels of the nodes define the underlying local structure for a particular class, but often this does not necessarily overlap significantly with the local structure defined by the the edge connectivity of the graph. Alternatively, realistic graphs typically have large-scale properties that are very poorly structured with respect to the behavior of random walks [152, 155, 153, 134, 10, 9].

The basic assumption of random walk based methods (as well as other related methods) is that nodes visited more often than others by random walks starting from a particular node are also more useful to describe that node in terms of downstream prediction tasks. However, the problem with random walks is that typically most of the graph can be reached within a few steps, and thus information about where the random walk began (which is the node for which these methods are computing the embedding) is quickly lost.

This issue is particularly problematic for extremely sparse graphs with upward-sloping Network Community Profiles (NCPs) [152, 155, 153] and for flat NCPs [134] (expander-like graphs) or deep k-cores [10, 9]. These properties are ubiquitous among realistic social and information networks. This suggests that, unless carefully engineered, embedding methods based on random walks will perform sub-optimally, since the random walks will mix rapidly, thereby degrading the local information that one hopes they identify.

In this chapter, we explore these issues, and we present a method which takes the local neighborhood structure of each node in the graph individually into account. This leads to improved embedding vectors and improved results in downstream applications for graphs.

Our method, LASAGNE, is an unsupervised algorithm for learning locality and structure aware graph node embeddings. It uses an *Approximate Personalized PageRank* vector [22] to adapt and improve state-of-the-art methods for determining the importance of the nodes in a graph from a specific node's point of view. The proposed methodology is easily parallelizable, even on distributed environments, and it has even been shown that the methods we adapt were applied to graphs with more than billions of nodes on a single machine [234].

We evaluate our algorithm with multi-label classification and link prediction on several real-world datasets from different domains under real-life conditions. Our evaluations show that our algorithm achieves better results—especially for downstream machine learning tasks whose objectives are sensitive to local information—in terms of prediction accuracy than the state-of-the-art methods, and our algorithm achieves similar results for link prediction. As has been described previously [155, 134, 10, 9], and as we review in Section 10.4, graphs with flat NCPs and many deep k-core nodes have local structure that is particularly difficult to identify and exploit. Importantly, our empirical results for this class of graphs is substantially improved, relative to previous methods. This illustrates that, by carefully engineering locally-biased information into node embeddings, one can obtain improved results even for this class of graphs, without sacrificing quality on other less poorly-structured graphs.

We also illustrate several reasons why random walk based methods do not perform as expected in practice, justifying our interpretation that our method leads to improved results due to the manner in which we engineer in locality.

The remainder of the paper is as follows: in Section 10.2, we survey related work, including the *word2vec* framework and the approximate computation of the *Personalized PageRank*; in Section 10.3, we describe our main LASAGNE algorithm; in Section 10.4, we present the evaluation of our method and a discussion of disadvantages of previous random walk based methods; and in Section 10.5, we present a brief conclusion.

10.2 Preliminaries

Embedding Words with Word2vec

Word2vec [182, 149] is a framework for learning word representations in some vector space by simultaneously preserving the words' semantic meaning. The representations are learned based on some contexts so that embeddings sharing similar contexts end up close to each other in the learned space. The embeddings are learned by maximizing the prediction probability of the contexts given the input embeddings, i.e., *Skip-gram* model. Note, that the model assumes independence of different contexts from each other for the same input. *Negative sampling* is used to estimate the prediction probability during the training. It maximizes the log probability of the input's context by simultaneously minimizing the prediction probability for k randomly selected contexts. Furthermore logistic regression is used to estimate the prediction probability:

$$\log \sigma(v_I'^T v_{c_i}) + \sum_{j=1}^{k} \mathbb{E}_{w_{j \sim P_n(w)}} \log \sigma(-v_I'^T v_j).$$

For each word the model maintains two representations, embedding and context representation. The vector v_I' denotes the embedding representation of the input, v_{c_i} is the context representation and v_j are representations of randomly selected contexts. The stochastic gradient descent algorithm is used for model optimization. The *Skip-gram* was later generalized to include arbitrary contexts [157].

Approximate Personalized PageRank

The *PageRank* algorithm [201] computes an "importance" score for every node in some graph. Each of the scores corresponds to the probability of a "random surfer" to visit a node given some start distribution. The *PageRank* vector is the solution of the linear system:

$$pr(s) = \alpha s + (1 - \alpha) pr(s) W, \tag{10.1}$$

with $W = D^{-1}A$ being the random walk transition matrix. A is the adjacency matrix, D is the degree matrix having the node degrees on the diagonal. The constant α is the *teleportation* probability. The starting nodes or more specifically the probability for each node to be the starting point of a random walk are given by the vector s. A variant of *PageRank* is the *Personalized PageRank* (PPR) whose result corresponds to the result of the *PageRank* algorithm, where the probabilities in the starting vector s are biased towards some set of nodes. The *push* algorithm described in [133] [33] [22] is used to compute an *Approximate Personalized PageRank* (APPR) vector in a more efficient way if the start distribution vector s is sparse, i.e., has probability mass on only a few nodes. The idea behind the *push* algorithm is to propagate a node's probability locally and only if there is a sufficient amount of probability to update. This leads to a sparse solution which means that only relatively few nodes of the underlying graph are contained in the resulting APPR vector.[1]

We describe the adapted version from [234] which converges faster. In addition to α and s, the main algorithm expects the approximation parameter

[1]We emphasize that this APPR method has been remarkably successful at characterizing the local and global structural properties in large social and information networks [152, 155, 153, 134], suggesting (as we show here) that it can also be used for improved supervised learning on these graphs.

ϵ. It maintains two vectors: the solution vector p and a residual vector r. The p vector is the current approximation of the PPR vector and vector r contains the approximation error or not yet distributed probability mass. In each iteration the main algorithm selects a node with sufficient probability mass in vector r and calls the *push* method. The probability mass from node's entry in r is spread between the node entry in p and the entries of its direct neighbors in r. The exact PPR is the linear combination of the current solution vector p and the PPR solution for r. The approximation quality and runtime are controlled by the parameter ϵ. The updates are performed as long as there is a node for which at least $\epsilon\frac{1-\alpha}{1+\alpha}$ probability mass is moved towards each of its neighbors during the *push* operation.

10.3 Lasagne: Locality And Structure Aware Graph Node Embedding

The Lasagne algorithm consists of two steps: a preprocessing step, which computes the APPR vectors for each node; and the learning step, which uses the APPR vectors to generate training examples batchwise to learn the final embeddings.[2]

Approximated Personalized PageRank for Node Embeddings

The computation of the APPR vectors for the node embeddings is described in Algorithm 5. There are two main modifications, relative to the original method in [234]. The first is the assignment of probability mass to the *seed node* in its own APPR vector, and the second is to the stopping criterion.[3]

The first modification allows the seed node to be considered as its own neighbor during sampling the training examples. Consequently, the seed node is considered to be similar to other nodes that have the seed node among their neighbors, which in turn leads to higher proximity of such nodes in the embedded space. To avoid each node being considered to be the most important member of its own neighborhood (and thus being overrepresented

[2]Both our approach and the previous approaches which we improve use some sort of random walk to construct “documents,” each “word” of which is a node from the graph, and then they call the *word2vec* method. Essentially, our two improvements use APPR to more precisely engineer in locality, thereby leading to higher-quality documents. The importance and sensitivity of such preprocessing is well known in natural language processing.

[3]These modifications seem minor, but getting them right is extremely important for obtaining a robust and successful method.

during the training phase), we replace the node's own entry in its APPR vector with the second highest probability, c.f. line 21 - 22.

The second modification is since our main motivation is not to approximate the PPR vector but instead to keep only the *relevant* neighbors that represent a meaningful context for the seed node. The algorithm avoids considering neighbors, each of which is visited relatively rarely by the random walk.[4] Thus, our algorithm stops when the new node, which can be added to the APPR vector during the next iteration has a low chance to be visited by the random walk compared to the overall probability of previously added nodes. The running time for the algorithm depends on the probability significance threshold δ. The number of updates of the APPR vector is at most $\frac{1}{\delta}$. Given that the amount of probability moved in subsequent steps is always lower, we can assume it to be the same. Therefore it holds that $sumProbUpdates = n \cdot probUpdate$, whit n being the number of previous steps. Given that $lastDistrUpdate = probUpdate/sumProbUpdates$, it follows that $lastDistrUpdate \leq \frac{1}{n}$.

Learning of Embeddings From Approximated Personalized PageRank Vector

The embedding learning process is described in Algorithm 6. Each training example is a pair of nodes. We call one of them *seed node* and the other one *neighbor node*. The embedding is learned for the *seed node* while the *neighbor node* is used as context. The embeddings are learned analogously to the *Skip-gram* model described in Section 10.2. For each training pair the probability of the *neighbor node* is maximized given the *seed node*.

To generate the training pairs, we sample the *neighbor nodes* based on the APPR vector of the corresponding *seed node*. This means that for each *seed node*, we consider only those nodes as context which have some probability mass in the *seed node's* APPR vector, i.e., relevant nodes. Each *neighbor node* is sampled with the probability proportionally to its entry in the *seed's* APPR vector. *Neighbor nodes* are sampled with replacement and the probability to be sampled is equal to the relative ratio of probability mass each *neighbor node* contributes to the entire APPR vector. With this sampling strategy training data can be generated on request and the number of training examples per node can be easily controlled. Using the alias method [143], the sampling setup costs are $O(k)$, where k is the size of the APPR vector and the costs to sample a *neighbor* are $O(1)$.

[4]These nodes tend to be "far from" the node of interest; but, in total, they may absorb a significant large amount of the overall probability mass.

Algorithm 5 LASAGNE ApproximatePPR

```
Input: Node $s$, teleportation parameter $\alpha$, Probability significance threshold $\delta$
Output: APPR vector p
 1: p = $\vec{0}$, r = $\vec{0}$, heap=heap()
 2: r(s) = 1
 3: heap.push((s,1))
 4: sumProbUpdates = 0
 5: lastDistrUpdate = 1
 6: while lastDistrUpdate > $\delta$ do
 7:     u = heap.pop()
 8:     probUpdate = $(2\alpha/(1+\alpha))$r(u)
 9:     if u $\neq$ s then
10:         sumProbUpdates += probUpdate
11:         lastDistrUpdate = probUpdate / sumProbUpdates
12:     end if
13:     p(u) = p(u) + probUpdate
14:     neighResUpdate = $((1-\alpha)/(1+\alpha))r(u)/d(u)$
15:     for v with (u,v)$\in E$ do
16:         r(v) = r(v) + neighResUpdate
17:         heap.update((v, r(v)/size(v.neighbours)))
18:     end for
19:     r(u) = 0
20: end while
21: p(s) = 0
22: p(s) = max(p)
23: return p
```

Parallelization

Our approach scales linearly with number of nodes and can easily be parallelized. The APPR vectors can be computed independently for each node and as shown in [234] even the largest publicly available graphs fit into the memory of todays commodity hardware. The learning procedure can be parallelized in two ways: the sampling from APPR can be done independently in parallel; and the actual learning of the embeddings can also be processed in parallel either asynchronously or synchronously on multi-core or distributed architectures. For details see [136].

10.4 Empirical results

In this section, we summarize our empirical results. We have evaluated the node embeddings produced by the LASAGNE algorithm by performing prediction tasks which aim at inferring node labels in multi-label classification

Algorithm 6 Learn Embeddings

```
Input: List with seed node and APPR vector pairs apprs, maxBatches, batchSize
 1: samplers = emptyList()
 2: for seedNode and currentAppr in apprs do
 3:     samplers.add((seedNode, createAliasSampler(currentAppr))
 4: end for
 5: while not converged and batchNumber < maxBatches do
 6:     currentBatch = emptyList()
 7:     for seedNode and s in samplers do
 8:         neighbors = s.sample(batchSize / size(samplers))
 9:         trainingExamples = createPairs(seedNode, neighbors)
10:         currentBatch.add(trainingExamples)
11:     end for
12:     permute (currentBatch)
13:     negativeSamplingGradientDescent(currentBatch)
14:     batchNumber++
15: end while
```

and link prediction scenarios. We have used a variety of real-world graph datasets from various domains, i.e., a biological network, social networks, and a collaboration network. Here, we compare our results against the state-of-the-art techniques *DeepWalk*, *node2vec* and *GraRep*. Note that we omit a comparison with the *LINE* since it is already shown in [103] and [62] that the results produced by *node2vec* and *GraRep* are superior to the ones produced by *LINE*. We have implemented *GraRep* using sparse matrix operations. Despite of this, we were not able to run it for larger graphs due to out of memory errors. We tested on a machine with 387GB RAM.

Datasets

Network	$\|V\|$	$\|E\|$	$\|\mathcal{L}\|$	$\overline{d}$	$\overline{C}$	D	$\overline{D}$	k_{max}	$P_{k_{max}}$	Description
PPI	3,890	38,739	50	9.959	0.146	8	3.095	30	0.028	biological network
BlogCatalog	10,312	333,983	39	32.388	0.463	5	2.382	115	0.043	social network
IMDb Germany	32,732	1,175,364	27	35.909	0.870	11	3.487	102	0.009	collaboration network
Flickr	80,513	5,899,882	195	73.279	0.165	6	2.901	551	0.018	social network

Table 10.1: Statistics of networks used for multi-label classification: number of nodes $|V|$, number of edges $|E|$, number of classes $|\mathcal{L}|$, average degree $\overline{d}$, average clustering coefficient $\overline{C}$, diameter D and average shortest path length $\overline{D}$, maximum k of $k-cores$ k_{max}, fraction $P_{k_{max}}$ of nodes in k_{max} $k-core$

We consider the following graph datasets from various domains with different sizes and number of classes.

- Protein-Protein Interactions (PPI) [55]: This is a subgraph of the PPI network for Homo Sapiens which is also used in [103]. The network consists of 3,890 nodes that represent proteins and 38,739 edges which represent the existence of interactions between the corresponding proteins. The 50 different labels represent biological states.

- BlogCatalog [245]: This is a social network graph where each of the 10,312 nodes corresponds to a user and the 333,983 edges represent the friendship relationships between bloggers. 39 different interest groups provide the labels. This network is used in both [103] and [214].

- IMDb Germany: This kind of artificial dataset is created from the IMDb movie database [127]. It consists of 32,732 nodes, 1,175,364 edges and 27 labels. Each node represents an actor/actress who played in a German movie. Edges connect actors/actresses that were in a cast together and the node labels represent the genres that the corresponding actor/actress played.

- Flickr [245]: The Flickr network is a social network graph with 80,513 nodes and 5,899,882 edges. Each node describes a user and the links represent friendships. The 195 given labels stem from different interest groups. This dataset is also used in [214].

Table 10.1 summarizes some statistics of these networks.

The selection of networks captures different structures, and we use Network Community Profile (NCP) plots from [152, 155, 153, 134] to analyze them. The NCP depicts the best “score" for different clusters in the graph as a function of their size. The cluster “score” is defined by *conductance*, i.e., the ratio of edges going out of a cluster to cluster internal edges. As can be seen in Figure 10.1, the IMDb Germany network has quite clear clusters of about 50 to 100 nodes. For each outgoing edge in the small clusters with near-minimum conductance value, there are about 800 internal edges. The three other datasets are not well separable.[5] The best cluster in the Flickr graph has a size of about 5000 nodes and only about 50 internal edges for each outgoing edge.

Following [10, 9], we also use *k-core* information to analyze graph properties. The *k-core* of a graph G is the maximal induced subgraph $H \subseteq G$ such that every node in H has a degree of at least k. Figure 10.2 shows size of *k-cores* for all k for all four datasets. We call a core “deep” if the

[5]In particular, the cluster quality is only slightly better than that of a randomly-rewired graph; Lasagne does particularly well for these graphs.

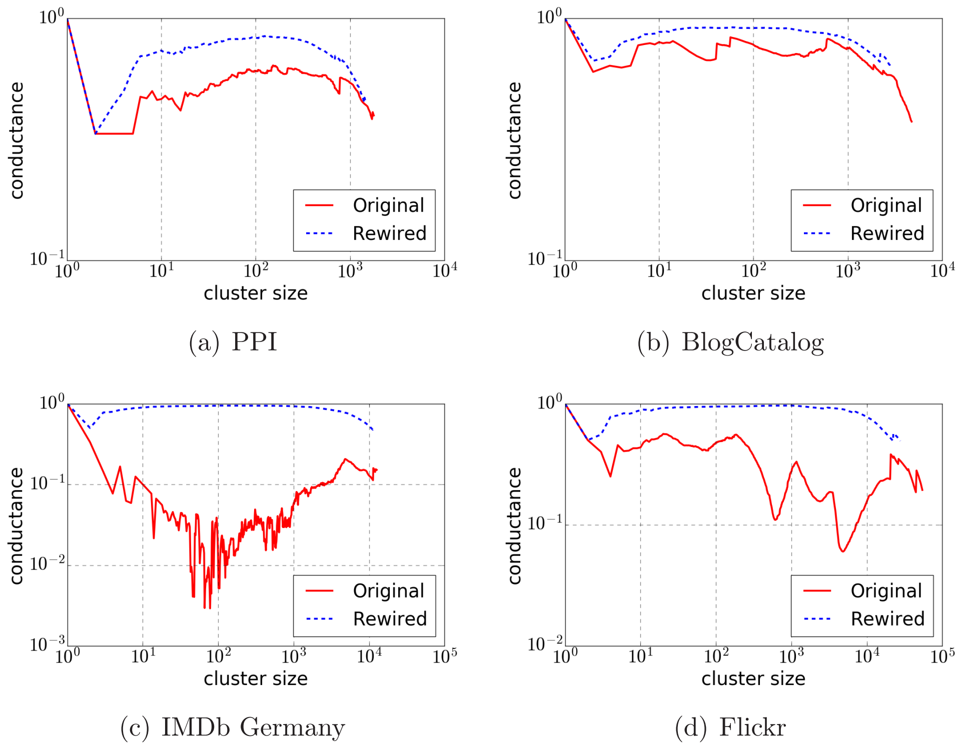
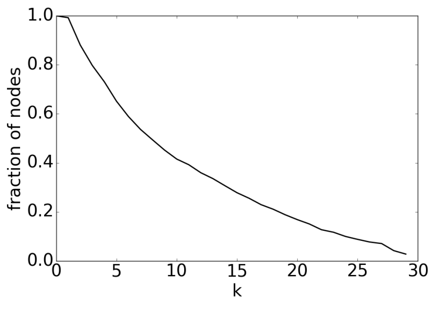


Figure 10.1: NCP plots for used datasets. Red, solid lines sketch the community structure of the original graph. (Down represents better cluster quality.) Blue, dashed lines plot the structure of randomly rewired networks.

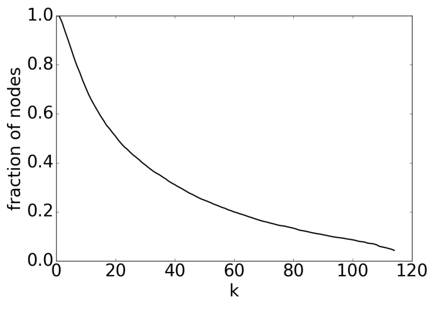
corresponding k is high. In 10.4 we discuss how size and depth of the k-*cores* affects the performance of different methods.

Experimental Setup

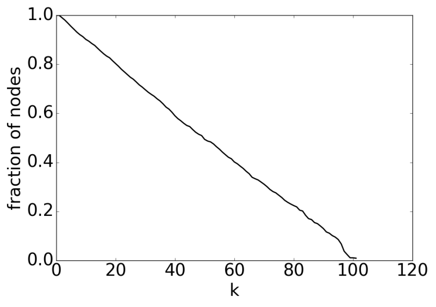
Like previous works, we use multi-label classification to evaluate the quality of the node embeddings. However, as discussed in the following, we think that the evaluation method for node representations used in [214, 244, 103] has a major drawback: it is hardly applicable in real world scenarios. Thus, we propose a new method for evaluating node embeddings that also relies on multi-label classification but is far closer to a real-life application scenario than the former method. We evaluate LASAGNE according to both evaluation metrics.



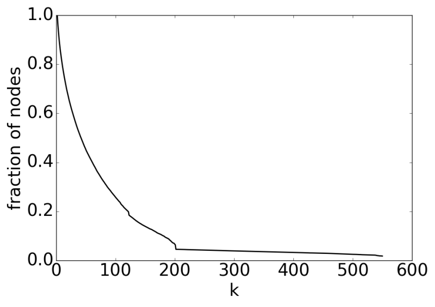
(a) PPI



(b) BlogCatalog



(c) IMDb Germany



(d) Flickr

Figure 10.2: k-core plots for used datasets. Note the different scaling on the x-axes.

Previous Method

Perozzi et al. [214] made the currently used evaluation method for graph node embeddings publicly available[1]. The procedure is as follows: a portion of the labeled vertices is sampled randomly and used as training instances, while the remaining nodes are used for testing. The sampling approach does not preserve the percentage of samples for each class, resp. labels. After sampling, one classifier is trained for each class by using one-vs-rest logistic regression and the labels for the test instances are predicted. For the actual prediction task, this method makes recourse to information that is typically unknown. Precisely, this method uses the actual number of labels k each test instance has. By sorting the predicted class probabilities and choosing the classes associated with the top k probabilities, prior knowledge is incorporated into the prediction task. In real world applications, it is fairly uncommon that users have such knowledge in advance. A label is considered

[1]https://github.com/phanein/deepwalk - last accessed: 2017-01-03

as a positive if it is among the top k predicted labels, regardless its real probability value. The entire evaluation procedure is repeated 10 times and finally the average macro-F_1 and micro-F_1 scores are calculated.

More realistic method

We propose the following modified evaluation metric that reflects better the real world classification scenario where no a priori knowledge is given. Generally, we also train logistic classifiers to predict the labels of the test instances. In contrast to the previous method, we suggest to use a 10-fold stratified cross-validation for each one-vs-rest classifier. Using such stratified sampling is a common way to split the data into training and test set by coincidently preserving the ratio of subpopulations within the data. In this way, the prediction accuracy does not suffer from classes that may not appear in either the training or the test set due to small numbers of positive examples. Furthermore, we get rid of using prior knowledge to determine the positive predicted labels. Instead of ranking the probabilities and taking the labels corresponding to the top k probabilities, we make the decision of labeling the test instance based on the label probabilities directly, i.e., if the probability of a label l is at least 50% we consider l as positive.

We use micro-F_1 and macro-F_1 as evaluation metrics. Macro-F_1 scores build the unweighted average of F_1 scores for positive classes over all classifiers. Micro-F_1 scores build the global average based on prescision and recall by treating each test example equally. We primarily focus the discussion on the macro-F_1 metric, but we also report the micro-F_1 scores.

Results of the More Realistic Evaluation Method

The results reported in this section were obtained by using the parameter settings suggested in [214]. We use $\gamma = 80$ as the length for the random walks performed during the *DeepWalk* and *node2vec* procedures.[6] The number of random walks is $|V| \cdot r$, with $|V|$ being the number of vertices and $r = 10$ being the number of random walks starting from each node in the graph. The size of the window which slides over each random walk sequence extends to at most $w = 10$ in each direction of the currently regarded vertex and the dimensionality of the node embeddings is set to $d = 128$. To get a fair comparison between our method and the random walk based methods, it is crucial to use similarly sized training sets for the learning procedure since larger training sets typically tend to result in higher prediction accuracy for

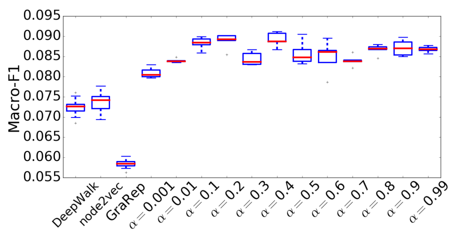
[6]If diameter $D = 5, 6, 8, 11$, then walk length $\gamma = 80$ is quite long.

the test phase. Thus we sample

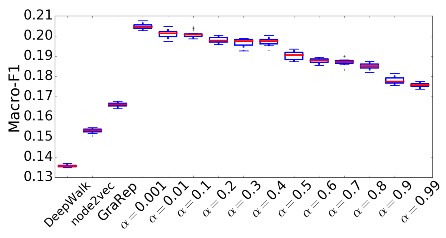
$$|T| = |V| \cdot \left[\gamma \cdot r \cdot 2 \cdot \mathbb{E}(\mathcal{U}(1,w)) - 2 \cdot \sum_{i=1}^{w} \mathbb{E}(\mathcal{U}(1,i))\right]$$

training examples which corresponds to the expected number of training instances generated by the random walk approaches. The notation $\mathbb{E}(\mathcal{U}(x,y))$ denotes the expected value of a uniform distribution $\mathcal{U}$ in the interval $[x,y]$.

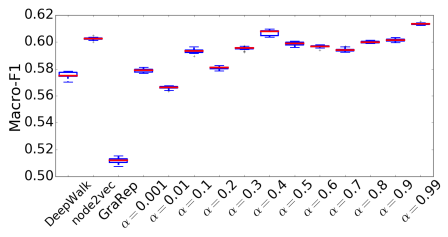
For *node2vec* we follow the suggestions of the authors and perform full grid searches over the set $\{0.25, 0.5, 1.0, 2.0, 4.0\}$ for both hyperparameters. The *GraRep* hyperparameter k is ranged from 1 to 6. For LASAGNE we used $\sigma = 0.0001$ as significance threshold for probability updates in all empirical evaluations. We show results for different values of teleportation parameter α. For all datasets and all approaches we demonstrate the results when we used 90% of the data for training and the remaining data as test set for the classification tasks. The distributions of resulting macro $F1$ scores are visualized as box plots. We adapted the computation of the *Approximated Personalized PageRank* implemented in the *Ligra* framework [233] for our implementation. The learning procedure for the embeddings is implemented in *TensorFlow* [1]. The code will be publicly available upon acceptance.



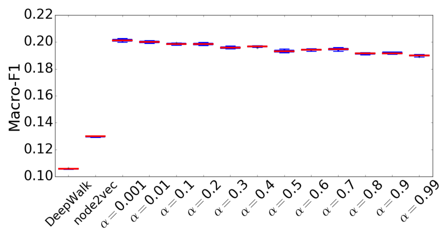
(a) PPI



(b) BlogCatalog



(c) IMDb Germany



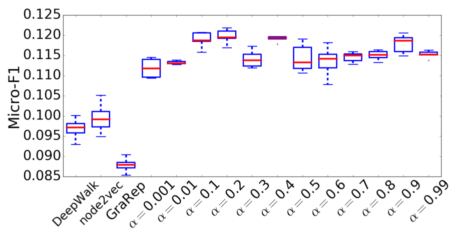
(d) Flickr

Figure 10.3: Macro-F_1 scores achieved by doing multi-label classification as downstream task for the considered representation learning techniques.

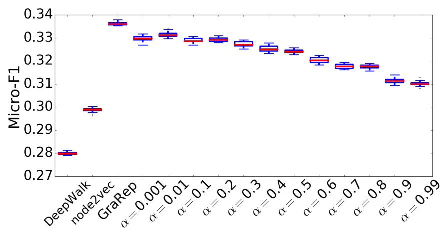
Figure 10.3 shows the macro-F_1 scores for all methods and datasets when applying embeddings for multi-label classification. LASAGNE overcomes the competitors for each dataset.

For the PPI network, c.f. Figure 10.3(a), the scores are steadily over 8% for all α values, while the random walk approaches reach scores between 7% and 7.5%. The best *node2vec* setting is $p = 4$ and $q = 1$, which corresponds to a rather low willingness to allow the random walks to return to already visited nodes. This meets the outcomes of LASAGNE, which are best for small α values. The generally low prediction quality for all approaches, and especially the bad score for *GraRep*, may indicate that the distribution of class labels do not follow any representative, local patterns and hence are hardly graspable within local structures (at least in this set of data). The results for BlogCatalog are even more clear. LASAGNE improves the best competitor by approximately 23%. As can be seen in Figure 10.3(b) the performance of LASAGNE decreases almost monotonically with increasing values for α. This means that the neighbors which describe a node best are not extremely local. The best *node2vec* setting, i.e., $p = 0.25$ and $q = 0.25$, confirms this results. Recalling Figure 2 from [103], the 2nd order random walks are biased towards leaving the neighborhoods. For IMDb Germany, c.f. Figure 10.3(c), the best result of LASAGNE, which is for $\alpha = 0.99$, is only slighty better than the best results achieved with *node2vec*. Since LASAGNE is, as well as *node2vec* with parameter setting $p = 0.25, q = 4$, able to stay extremely local, both approaches reach high prediction scores on this dataset where the labels are concentrated in low conductance clusters. Using the Flickr network, LASAGNE reaches the highest improvement over the other random walk based methods, i.e. more than 33%. The results behave similar to the ones for the BlogCatalog data, but in contrast the scores remain more stable. Indeed, the drop between the smallest and largest selected α values is only 1%. As mentioned previously, we could not run *GraRep* on Flickr, because of its size.

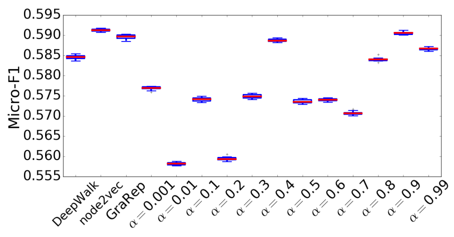
Figure 10.4 shows the micro-F_1 scores achieved with the same settings as used for the macro-F_1 score evaluation. The results show that the micro scores are higher than the macro scores for all datasets except for IMDb Germany. Also the relative differences between the results for LASAGNE and the best competitor are higher for the macro-F_1 scores than for the micro-F_1 scores. This is due to the micro score metric effectively gives higher weight to larger classes. This may be justified by the results depicted in Figure 10.7 (discussed below). Since LASAGNE performs better for smaller classes which, except for IMDb Germany, are the vast majority of classes, the macro-F_1 scores take benefit due to weighting each class equally independent from the class sizes. Recalling that the micro-F_1 considers the sizes of the classes,



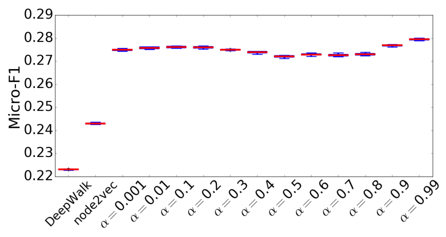
(a) PPI



(b) BlogCatalog



(c) IMDb Germany



(d) Flickr

Figure 10.4: Micro-F_1 scores achieved by doing multi-label classification as downstream task for the considered representation learning techniques.

the performance improvements for this score are reasoned by the fact that Lasagne performs better on smaller classes and similarly good to random walk based methodologies on larger classes.

An important summary point from Figures 10.3 and 10.4 is that, in the case of graphs without even small-sized good conductance clusters, the performance of Lasagne clearly overcomes the performance from random walk based methods. On the other hand, for graphs that have an upward-sloping NCP and thus small-sized good conductance clusters, Lasagne shows similar prediction quality to random walk based methods. In particular, while we are never worse than previous methods, we observe the weakest improvement for IMDb Germany, which is consistent with Figure 10.1(c), where the upward-sloping NCP suggests relatively good local structure, and we observe the strongest improvement for the Flickr network, which is consistent with Figures 10.1(d) and 10.2(d), which indicate a relatively flat NCP and many deep k-core nodes.

Results of the Former Evaluation Method

Tables 10.2 and 10.3 show the macro-F_1 scores, resp. the micro-F_1 scores when applying the evaluation proposed by [214] and using 90% of the node

Algorithm	Dataset			
	PPI	BlogCatalog	IMDb Ger	Flickr
DeepWalk	0.1747	0.2221	0.6868	0.2104
node2vec	0.1930	0.2418	0.6996	0.2349
GraRep	**0.1991**	0.2231	0.5770	-
Lasagne	0.1835	**0.2843**	**0.7042**	**0.2930**

Table 10.2: Macro-F_1 scores for multi-label classification when using former evaluation method and 90% of instances for training.

Algorithm	Dataset			
	PPI	BlogCatalog	IMDb Ger	Flickr
DeepWalk	0.2206	0.3889	0.7043	0.3762
node2vec	0.2293	0.3963	**0.7060**	0.3841
GraRep	**0.2487**	0.3913	0.6648	-
Lasagne	0.2216	**0.4116**	0.6967	**0.4078**

Table 10.3: Micro-F_1 scores for multi-label classification when using former evaluation method and 90% of instances for training.

representations for training. While *GraRep* shows the best results on PPI, the performance of the Lasagne embeddings clearly overcomes the competitors when testing on the considered social networks, similar to the results in our more realistic (and more refined) evaluation.

Link Prediction

For completeness, Table 10.4 reports the results when applying the embeddings retrieved by Lasagne on the link prediciton task. The experimental setup is borrowed from [103] which means that we removed 50% of the edges of each graph, learned the representations on the remaining graph and finally predict the existence of the removed edges by using a binary classifier. The classifier is trained with the remaining 50% of edges as positive examples and the same amount of non-existent edges as negative samples. The edges were embedded by using one of the embedding methods documented in Table 10.4. Hence, an edge embedding is the combination of the representation of the nodes joined by the corresponding edge according to the specified method. As evaluation metric we also use the well-known Area Under the Curve (AUC) score. For Lasagne and *node2vec* we used the same set of parameter settings as for multi-label classification. The reported results are the best results that were achieved by all settings. We consider the following graphs for link prediction:

- Facebook [154]: This is a social network consisting of friend lists from facebook. The network consists of 4,039 nodes that represent users and 88,234 edges which represent friendships between the corresponding users.

- BlogCatalog [245]: This is the same network as in Table 10.1.

- arXiv Astro-Ph [154]: This is a collaboration network which covers scientific collaborations. It consists of 18,772 nodes and 198,110 edges. Each node represents an author and edges connect authors who collaborated on a joint work submitted to arXiv astro physics category.

Facebook (which has a relatively flat NCP [155, 134]) and arXiv Astro-Ph (which has an upward-sloping NCP [155, 134]) were also used in *node2vec* [103] for link prediction.
Overall, these results show that the Lasagne embeddings perform as well as the representations learned by *node2vec* when considering the facebook dataset or the arXiv dataset. The actual differences between the best results are less than 1%. For the BlogCatalog data, the representations retrieved by Lasagne even improve the best prediction score reached by the random walk based competitors. Disregarding the edge embedding methods proposed in [103] and using the jaccard similarity (jac), i.e., $jac(u, v) = \frac{\mathcal{N}_k(u) \cap \mathcal{N}_k(v)}{\mathcal{N}_k(u) \cup \mathcal{N}_k(v)}$ with $\mathcal{N}_k(u)$ being the k nearest neighbors of node u in the embedded space, instead, Lasagne also shows similar results as the competitors and yields the best results on the facebook data.

Explaining our improved empirical results

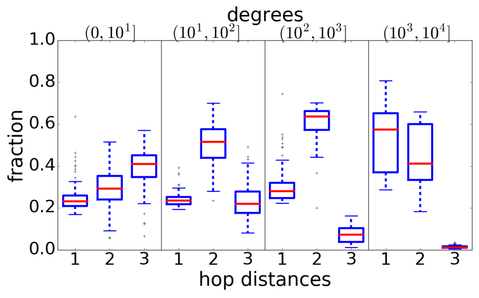
In this section, we present additional empirical results aimed at explaining in terms of graph locality properties Lasagne's improved performance. Lasagne improves previous methods by considering more finely the structure of the graph around each node. In particular, we compute local node neighborhood by touching only the *relevant* neighbors of each node, which leaves the major part of the graph unconsidered. For the node a we call b its *relevant* neighbor if b has high probability to be visited by random walk with restart starting from a.

Locality for nodes with different degrees

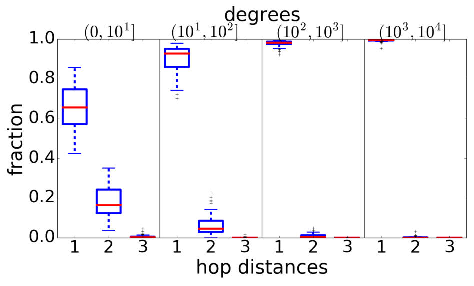
Previous random walk based methods follow a similar scheme, except they simulate *long* random walks in the graph. For each node occurring in one of the random walks, a window of dynamic size which contains nodes visited previously and after that node is used to determine the context. The actual extension of the window to each side is sampled each time uniformly from the interval $[1, w]$, where w is a hyperparameter (that is the same for all nodes in the graph). For example, while simulating the random walks the *DeepWalk* [214] algorithm selects the next node fully arbitrary among the neighbors of the last visited node. *node2vec* [103] generally gives more control over the context selection due to its hyperparameters and thus allows the prioritization of closer resp. farther neighbors. This flexibility comes to the cost of an expensive preprocessing step which is quadratic in node degree.

Nevertheless, even with this expensive preprocessing, existing methods fail to adapt to the local graph structure. When random walks are used to obtain neighbors, nodes having very low probability to be visited also appear among the considered neighbors. Nodes having high probabilities to be visited appear more frequently. However, the cumulative probability of low probability nodes may still be significant. The wider the window is, the more far away neighbors end up in it. However, smaller window sizes will not help to tackle the problem with low probability neighbors, since the nodes in sparse graph areas may have distant neighbors with high probability to be visited by random walk. Grover et al. [103] even show, that they achieve better results with larger window sizes. However, since the same window size is used for all nodes in the same graph, the distributions of hop distances of nodes to their neighbors are similar and barely adapt to local node neighborhood.

To confirm this intuition, we computed the hop distances to the nodes considered as context by *node2vec* and *DeepWalk* algorithms for different datasets. For all of them, we observed similar behavior, i.e., the level of locality was barely adapted with increasing node degree, c.f., Figure 10.5(a). Note that the *node2vec* parameters were set to $p = 0.25$ and $q = 4.0$, which constrains the random walks to capture very local neighborhoods (but in a non-adaptive manner). The distributions of hop distances to the neighbors found by the LASAGNE algorithm are very similar per dataset; an example is depicted in Figure 10.5(b). In contrast to the previous methods, LASAGNE adapts to the local node environment, i.e., for the high degree nodes only the neighbors with the highest probability to be visited by the random walk are considered as context. Consequently, we observe a clear tendency that the preference to local neighborhoods increases with increasing node degree



(a) *node2vec*



(b) LASAGNE

Figure 10.5: Distributions of hop distances to neighbors from nodes with different degrees. These plots visualize the ability to adjust to differently dense areas in the graph for *node2vec* (left, not well) and LASAGNE (right, very well).

(which is known to correlate with poor NCP clusters and deep k cores [155, 134, 10]). The *LINE* algorithm considers only one hop neighbors, and the assumption that only direct neighbors are relevant is very strong, especially for low degree nodes.

Locality for more versus less peripheral classes

Large graphs with flat NCP, especially with large and highly connected regions (with large deep *k-cores*) are notably affected by random walk problems. For graphs with flat NCPs, the connectivity among nodes' *relevant* neighbors is not much stronger than to the rest of the graph. Furthermore, the larger and deeper are graphs *k-cores*, the more time random walks will spend in them. This affects the neighborhoods obtained by random walks for most nodes, since most parts of even large graphs can be reached within few steps. Therefore, even if dense parts of the graph have high probabilities to be visited by global random walks, if the probabilities of single nodes in these components are low, then nodes from these components are not considered by LASAGNE as neighbors. Consequently, for the nodes from large deep *k-cores*, the neighborhood will be restricted to the most *relevant* core neighbors. Therefore, our method adapts to the structure of local neighborhood.

To confirm this intuition, we used the Flickr network, a graph with flat NCP. Figure 10.2(d) shows the fraction of nodes in different *k-cores* of this graph. As can be seen in Figure10.2(d), the graph has large deep *k-cores*, e.g., about 30% of nodes are in the subgraph where each node has degree 100 or more. We expect random walk based methods to perform poorly on such a graph, especially if the similarity to neighbors outside of large deep *k-cores* is important for the downstream task.

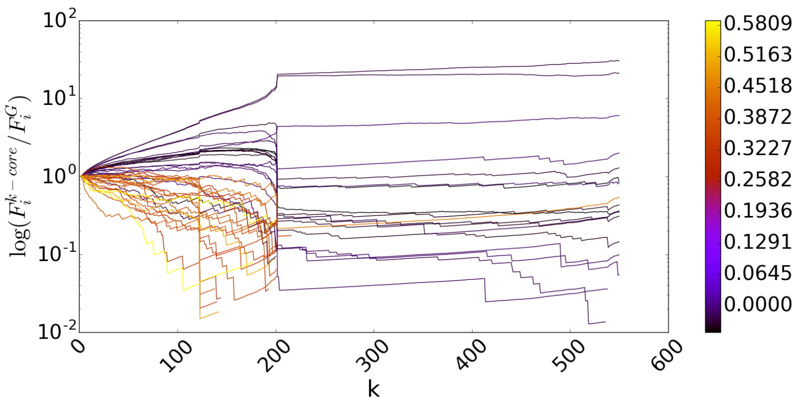


Figure 10.6: Each line depicts the class label distribution in k-cores with performance information for one class in the Flickr data. X-axis: k-core; Y-axis: log scaled proportion between fraction of class label i within the k-core, i.e., F_i^{k-core}, and the fraction of this class label within the entire graph G, i.e., F_i^G; color code: absolute difference in F_1 score between LASAGNE and the best random walk based method. For ease of presentation, the plot shows only the 20 classes where LASAGNE reached the highest improvement as well as the 20 classes where the improvement was smallest.

As multi-label classification is a common downstream task, Figure 10.6 provides empirical evidence that LASAGNE's embeddings overcome performance issues of previous embeddings. In Figure 10.6, each line stands for a class, and the color depicts the classification improvement of LASAGNE over best previous method. Additionally, the plotted line shows the fraction of nodes with the corresponding class label in each *k-core*, relative to the fraction of nodes with that label in the entire graph. When the fraction of class labels is zero, the line breaks. It can be clearly seen from the plot that LASAGNE achieves the best improvement for classes with members outside of large *k-cores* with high k, i.e., for classes that are more peripheral.

Relatedly, we also expect that our method has better performance on the tasks which require very accurate determination of local neighborhoods. Small classes (in particular) need this, since nodes of such classes are very sensitive to irrelevant neighbors. This is due to the small number of nodes that belong to the same class. Figure 10.7 shows plots which visualize the improvement of LASAGNE over random walk approaches for single classes. The improvements tend to be especially notable for the small classes, which confirms our claim.

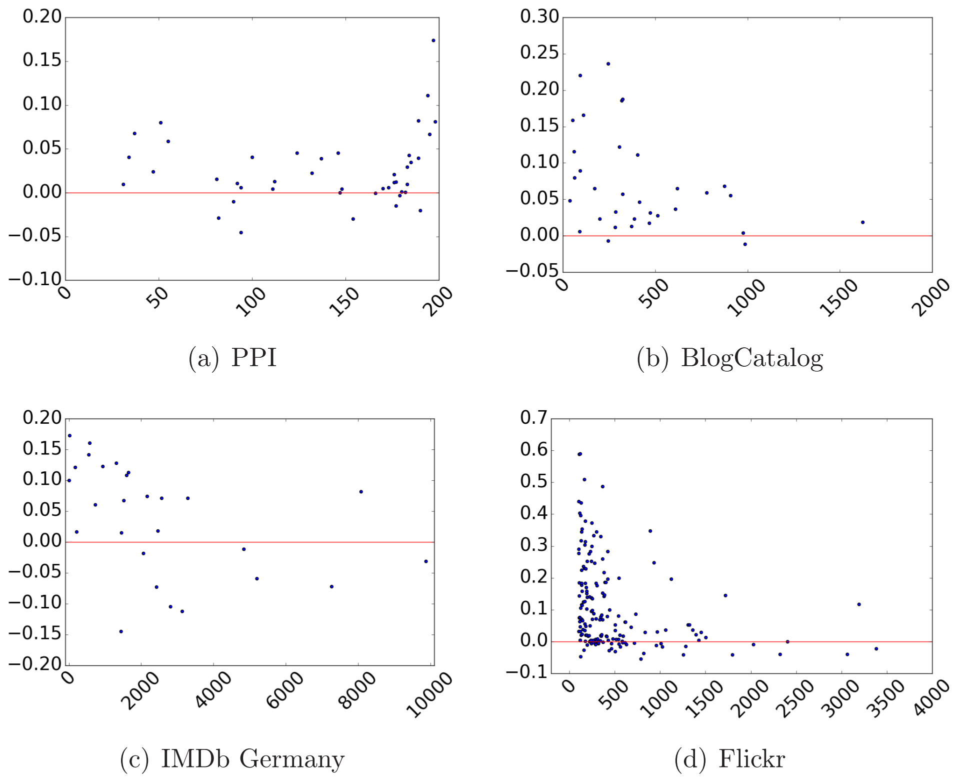
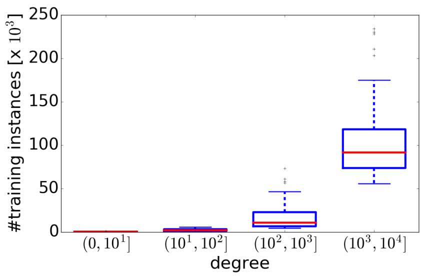


Figure 10.7: Absolute differences in F_1 scores between LASAGNE and the best random walk based method for single classes. One dot stands for one class. X-axes show the class size; Y-axes show the absolute difference in F_1 scores.

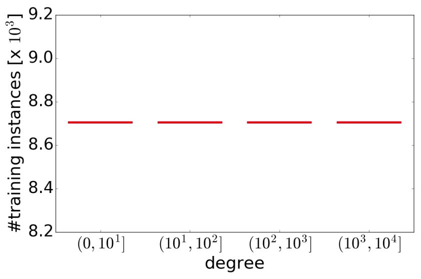
Distribution of training examples per node

Another shortcoming of existing random walk based methods is the distribution of training examples per node that they generate. Since high-degree nodes are visited more often by random walks, there are more training examples for them. Since small-degree nodes are visited much less often, they are underrepresented during training. Due to the way in which locality is engineered into LASAGNE, it solves this problem.

To confirm this intuition, we run random walk based algorithms for different datasets and counted the number of training examples for a sample of nodes with different degrees. For each degree range 100 nodes were randomly sampled. For all datasets we observed very similar distributions, also with different *node2vec* parameters. An example is shown in Figure 10.8(a). As can be seen, the number of training examples for previous methods still strongly depends on node degrees. In contrast, when using LASAGNE, the



(a) *node2vec*



(b) LASAGNE

Figure 10.8: The number of considered training instances for nodes of different degrees for *node2vec* (left) and LASAGNE (right). LASAGNE allows us to control the number of training instances per node; we used 8700 for this plot.

number of training examples per node can easily be controlled, in this case to be uniform, which prevents us from generating extremely unbalanced training sets, c.f. Figure 10.8(b).

10.5 Conclusion

We have proposed LASAGNE, an unsupervised learning algorithm to compute embeddings for the nodes of a graph. The basic idea of LASAGNE is to use an *Approximate Personalized PageRank* algorithm to bias random walks more strongly to the local neighborhood of each node; and, thus, the embedding for a given node is more finely tuned to the local graph structure around that node than the embeddings from previous similar methods. Our method performs particularly well for larger graphs that are not well-structured, e.g., that have flat NCPs and/or have many nodes in deep k-cores. Our empirical evaluation has shown that our embeddings achieve superior prediction accuracy over competitors when used for multi-label classification in several different real-world networks. Our empirical results also provide evidence justifying the reason for this improvement. While LASAGNE is primarily an exploratory tool, if one wants to use it in a more automated manner, then an important question will be how to automate the averaging of the APPR vectors over different values of the locality parameter.

op	Algorithm	Dataset		
		facebook	arXiv	BlogCatalog
a)	DeepWalk	0.7240	0.7002	0.7921
	node2vec	0.7223	0.7259	0.8108
	GraRep	0.7495	0.7097	0.8759
	LASAGNE	0.7069	0.7195	0.8701
b)	DeepWalk	0.9610	0.8632	0.7187
	node2vec	0.9644	0.8770	0.7359
	GraRep	0.9629	0.7494	0.8846
	LASAGNE	0.9628	0.8715	0.8281
c)	DeepWalk	0.9606	0.8438	0.7799
	node2vec	0.9642	0.8499	0.8044
	GraRep	0.9621	0.7980	0.8713
	LASAGNE	0.9072	0.7036	0.7017
d)	DeepWalk	0.9593	0.8450	0.7844
	node2vec	0.9646	0.8523	0.8074
	GraRep	0.9635	0.7664	0.8731
	LASAGNE	0.9111	0.7053	0.7045
jac	DeepWalk	0.8435	0.7357	0.5525
	node2vec	0.8509	0.7381	0.5644
	GraRep	0.8418	0.4980	0.5567
	LASAGNE	0.9256	0.7361	0.5337

Table 10.4: Results for Link Prediciton; Metric: AUC scores of predictions retrieved by binary classifiers resp. Jaccard similarity measure; Operators used for edge embedding: a) Average: $\frac{f_i(u)+f_i(v)}{2}$, b) Hadamard: $f_i(u) \cdot f_i(v)$, c) Weighted L1: $|f_i(u) - f_i(v)|$, d) Weighted L2: $|f_i(u) - f_i(v)|^2$, with $f_i(x)$ being the i-th component of node x [103]; jac: Jaccard similarity measure

Chapter 11

Structure-Based Node Embedding

Parts of the work presented in this chapter has been published as the article *Structural Graph Representations based on Multiscale Local Network Topologies* in the Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2019 [46]. A preliminary work-in-progress version has been published as the short paper article *Towards Learning Structural Node Embeddings using Personalized PageRank* in the Proceedings of the LWDA, 2017 [47].

11.1 Introduction

The increasing relevance of graph-structured data has been accompanied by an increased interest in algorithms which can leverage underlying graph structure to make accurate predictions about the modeled entities. In many scenarios no additional facts about entities or properties of relationships are known. In such cases, the only source of information for machine learning tasks like node and graph classification is the graph topology. In this chapter, we consider the problem of deriving structural node representations or *role representations* based solely on the topological structure within the local node neighborhoods.

The intuition behind this task is to capture the different functionalities of network entities within vector representations such that entities that play a similar role within the network end up close together in the embedded space. Note that the notion of a *role* is generally diverse and might describe influencers in a social network, or a specific group of atoms in molecule networks that are likely to bind to similar atomic substructures. In general, node representations describing the roles of the nodes within a graph are useful for downstream classification or clustering tasks, e.g., they may give

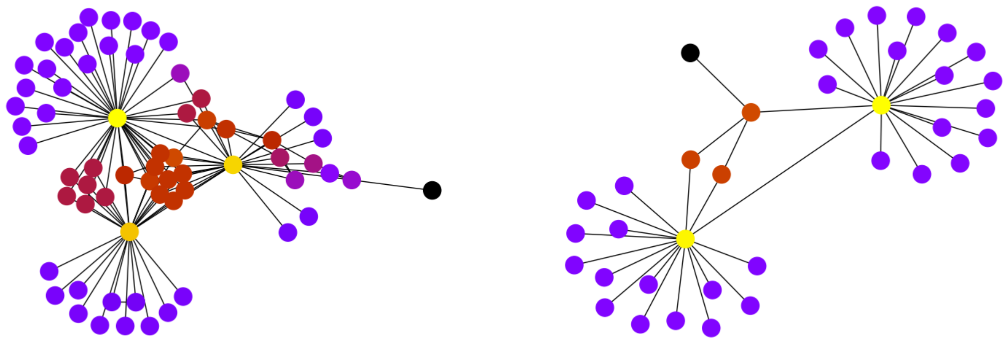


Figure 11.1: Airline networks. Left: Scandinavian Airline; Right: Niki Air. Each node corresponds to an airport and edges connect two airports if the airline operates a flight between them. The color coding corresponds to the role descriptors as determined by our approach.

valuable insights for real world tasks like drug design, identification of influencers within social communities, link prediction, etc. To introduce the problem more formally, we are given a set of graphs $\mathcal{G} = \{G_1, \ldots, G_N\}$ with $G_i = (V_i, E_i)$ being a graph, $V = \bigcup_{i=1}^{N} V_i$ denoting the set of vertices and $E = \bigcup_{i=1}^{N} E_i$ being the set of edges. The goal is to derive vector representations $f(v_j) \in \mathbb{R}^d$ reflecting the various roles of nodes $v_j \in V$ in a graph $G_i \in \mathcal{G}$. Figure 11.1 illustrates the concept of node roles for airline networks. Note that the color coding corresponds to the nodes' roles as identified by our approach. It can clearly be seen that the role of a node (e.g. hub airport, airport with connection to a single hub or an airport with connection to several hubs) can be extracted from the node's local neighborhood. We can also see that these roles can be identified across different graphs although the local neighborhoods may seem to be different, e.g., in terms of size. However, considering the local topology around the nodes, it can also be seen that they are similar for those nodes that have similar colors. The distinct property of node role representations is that they should be independent of specific neighbors. Therefore, nodes that are similar in the embedded space are not necessarily closely connected and even may reside in different graphs. In general, these representations could be either continuous vectors, such as structural node embeddings, or discrete role assignments, and should capture structural properties of the nodes within the graph. However, here, we focus on continuous representations. In particular, given two nodes $u, v \in V$ which have similar local structural neighborhood patterns with respect to some similarity measure, i.e., $S_N(u) \approx S_N(v)$, then the representation of u and v shall be similar as well. However, defining an appropriate similarity measure seems difficult since even the notion of local structural neighborhood patterns is hard to grasp. In this chapter, we argue that the spread of probability mass under the node's most relevant local neighbors is a good characteristic for the node's role. Similarly to [91] we leverage the *Approximate Personalized PageRank (APPR)* to effectively describe multiple locality structures around the vertices and use the probability distribution vectors as a basis to quantify the structural roles of the nodes. An important feature of our novel node representation is that it is very efficient to compute and thus, even suitable for large data sets. Furthermore, an important difference to previously published related works, e.g., [82], is that our method operates directly in the vertex domain, though the heat kernel diffusion process resembles that implied by PPR [71]. Additionally, our method is not restricted to k-hop neighborhoods.

Our empirical evaluation demonstrates that our simple approach outperforms somewhat more advanced state-of-the-art role-based node representations. With respect to previously published work on the topic of structural node embeddings (see Section 9.1) we summarize the key contributions of the work presented in this chapter as follows:

- A novel structure-based approach to determine role representations for single nodes directly in the vertex domain as opposed to existing diffusion-based approaches which operate in the spectral domain.
- A fast-to-compute approach that retrieves continuous role representations rather than being composed of multiple, computationally rather costly structural features.
- An extensive evaluation of our proposed role representations that shows promising results when comparing our representations to state-of-the-art node embeddings when using their setups.

11.2 Structural Node Representations using Approximate Personalized PageRank

As we use the Approximate Personalized PageRank to derive our role descriptors, we start this section with a short review of this method. Note that this is a slightly different version than the one presented in the previous chapter since we omit the adaptation to cut the long tails from the stationary distributions here. Afterwards, we present our method for extracting structural node descriptors. In Chapter 12, we finally describe how they can be aggregated to derive descriptors for whole graphs.

Algorithm 7 APPR

```
Input: Source node $v_i$, Teleportation probability $\alpha$, Approximation threshold $\epsilon$
Output: APPR vector $p_i$
1: $p_i = 0$, $r_i = e_i$
2: while $r_{ij} \geq \epsilon d_j$ for some vertex $v_j$ do
3:     pick any $v_j$ where $r_{ij} \geq \epsilon d_j$
4:     push($v_j$)
5: end while
6: return $p_i$
```

Algorithm 8 push(v_j)

```
1: $p_{ij} = p_{ij} + (2\alpha/(1+\alpha))r_{ij}$
2: for $v_k$ with $(v_j, v_k) \in E$ do
3:     $r_{ik} = r_{ik} + ((1-\alpha)/(1+\alpha))r_{ij}/d_j$
4: end for
5: $r_{ij} = 0$
```

Approximate Personalized PageRank

Personalized PageRank (PPR) can be viewed as a special case of the *PageRank* [201] algorithm, where each random walk starts at the same node v_i and at each step there is a chance of α to jump back to v_i. The effect of this modification is that the PageRanks are personalized to the node v_i, i.e., they represent the importance of each node from the perspective of the source node v_i. Formally, the PPR-vector π_i of node v_i is given as the solution of the linear system

$$\pi_i = \alpha e_i + (1-\alpha)\pi_i W, \tag{11.1}$$

where $W = D^{-1}A$ is the random walk transition matrix obtained from the $n \times n$ adjacency matrix A by normalizing the outgoing links of each node by its out-degree, $e_i \in \mathbb{R}^{1\times n}$ denotes the i-th unit vector and α is the teleportation parameter.

The probability of transitioning to a neighbor v_j from a node v_i is given by w_{ij}. The entry π_{ij} can then be interpreted as the probability that a random walk starting at v_i and stops at v_j. The expected length of a random walk is determined by the teleportation probability α. With a smaller value, a larger portion of the graph is explored, while a larger value leads to stronger localization.

Intuitively, π_{ij} measures the importance of node v_j for node v_i and the PPR vector π_i as a whole yields a distribution of the node importance in the neighborhood of v_i where the extension of the neighborhood is controlled by the parameter α. In particular, the neighborhood is not restricted to nodes with a maximum hop-distance, such as the k-neighborhood, which may

Algorithm 9 APPR-Roles

```
Input: Graph $\mathcal{G}$, Labels $\mathscr{L}$, Teleportation probabilities $\alpha$s, Approximation threshold $\epsilon$
Output: Classification model $m$
 1: $role\_descriptors$ = list()
 2: for $idx$ in range($\mathcal{G}$) do
 3:     v = $\mathcal{G}$.getNode($idx$)
 4:     $emb_v$ = list()
 5:     for $\alpha$ in $\alpha$s do
 6:         $p_v^\alpha$ = APPR($v, \alpha, \epsilon$)
 7:         $emb_v$.append($entropy(p_v^\alpha)$)
 8:     end for
 9:     $role\_descriptors$.append($emb_v$)
10: end for
11: $m$ = LogisticRegression().fit($role\_descriptors$, $\mathscr{L}$)
12: return $m$
```

contain irrelevant nodes or miss important ones. Compared to the shortest path distance, nodes with a larger shortest path distance from v_i could still be more important, e.g., if they can be reached via many different short paths. For similar reasons, nodes with a small shortest path distance might not be equally important. Such effects are captured by PPR.

Local push-based algorithms [133, 33] compute *Approximate Personalized PageRank (APPR)* very efficiently and lead to sparse solutions, where only the most relevant neighbors are taken into consideration [91]. In addition to the teleportation parameter α, the approximation threshold ϵ controls the approximation quality and runtime. The main idea is to start with all probability mass concentrated in the source node and then repeatedly push probability mass to neighboring nodes as long as the amount of mass pushed is large enough. In this work, we consider the algorithm proposed in [22]. In particular, we use an adapted version proposed in [234] which converges faster. The procedure is formalized in Algorithm 7 and Algorithm 8.

For a given graph G_i and teleportation probability α, we compute the APPR-vector $p_j^{(\alpha)}$ of each node $v_j \in V_i$ and store it as the j-th row of the sparse $n \times n$ APPR-matrix $P_i^{(\alpha)}$.

Entropy-based Node Descriptors

The APPR-vector p_i of a node v_i effectively models the connectivity of that node with respect to all other nodes in the graph as a probability distribution, where the probability mass is concentrated only on v_i's relevant neighbors. This way, the size of the neighborhood is determined by the parameter α.

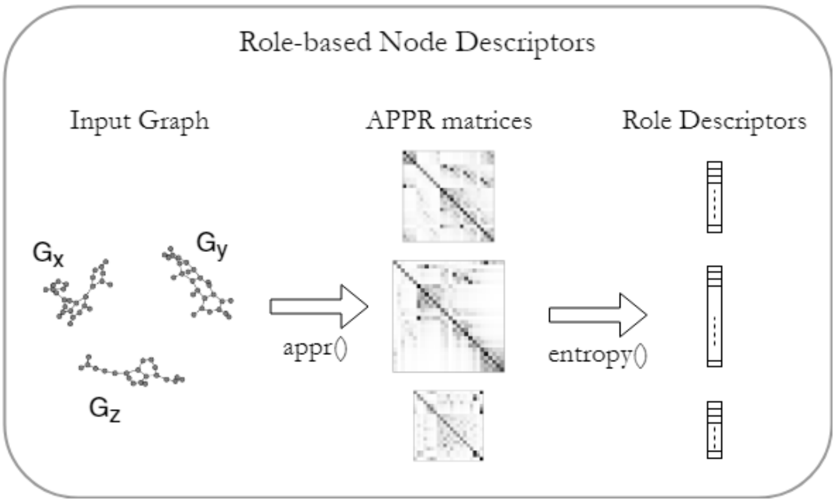


Figure 11.2: Workflow for calculating the role-based node descriptors.

The APPR-vector p_i additionally focuses on the most relevant neighbors by ignoring nodes with small probabilities and thus providing a sparse neighborhood representation. In principle, we could use the APPR-vectors directly as node representations. This would lead to the following feature space:

$$\Delta^n = \left\{ p \in \mathbb{R}^n_{\geq 0} \;\middle|\; \sum_{i=1}^{n} p_i = 1 \right\}, \tag{11.2}$$

which is known as the n-dimensional standard simplex. However, the resulting representations model homophily rather than structural properties, since they encode the information to which individual nodes a particular source node is connected. In order to make the representations location-invariant, we need to factor out this information. Since location invariance in this case translates to permutation invariance, we consider the quotient space

$$\Delta^n/_{\sim} = \{[p] \mid p \in \Delta^n\}, \tag{11.3}$$

which corresponds to the set of equivalence classes $[p] = \{q \in \Delta^n \mid p \sim q\}$ of the equivalence relation $\sim$ with $p \sim q \Leftrightarrow \exists P \in \mathbb{P} : p = qP$ where $\mathbb{P} = \left\{P \in \{0,1\}^{n \times n} \;\middle|\; P1 = 1,\ P^T 1 = 1\right\}$ is the set of permutation matrices. As a corresponding quotient map, we can define $f : \Delta^n \to \Delta^n/_{\sim}$ with $f(p) = pP_p$ which maps p to its equivalence class by sorting it using the permutation matrix P_p such that $1 \geq f(p)_1 \geq \cdots \geq f(p)_n \geq 0$.

Though the resulting sorted APPR-vectors qualify as structural node descriptors, they are not well suited for further downstream tasks since they are high-dimensional and sparse. Furthermore, node descriptors would not be comparable among graphs with different numbers of nodes. To this end we need to perform some form of aggregation. Our approach is based on the observation that, in terms of APPR, the structural properties of two nodes differ mostly based on the extent to which they spread their probability mass throughout the graph. For instance, a community node will spread its probability mass evenly to nodes within the same community, whereas a peripheral node will strongly concentrate its probability mass to one or very few nodes to which it is connected. The above behavior can be accurately described by the *Shannon entropy* $H : \Delta^n/\sim \, \to \mathbb{R}$ with $H(p) = -\sum_{i=1}^{n} p_i \log p_i$ where we use the binary logarithm. In particular, it fulfills the following properties.

Theorem 1. *For all* $p \in \Delta^n/\sim$ *it holds that*

1. $H(p) \in [0, \log n]$.
2. $H(p) = 0$ *if and only if* $p = e_1$.
3. $H(p) = \log n$ *if and only if* $p = \frac{1}{n}1$.
4. $H(p) = \log n - D_{KL}(p \| \frac{1}{n}1)$.

Proof. The proofs are straightforward and can be found in [74]. □

Intuitively stated, properties (1) to (3) state that the entropy is minimized for a distribution with a single peak and maximized for the uniform distribution. Property (4) states that the entropy can be interpreted as the similarity to the uniform distribution in terms of the *Kullback-Leibler divergence*. Our empirical results support the usefulness of this intuition. A further advantage over other applicable dimension reduction techniques is that we can describe each node by a single scalar value which can be visualized directly on a color map (as was done in Figure 11.1) and has a simple and intuitive meaning. Note that the entropy function is symmetric, i.e., $H(f(p)) = H(p)$ for all $p \in \Delta^n$. As a result, the APPR-vectors need not be sorted and the entropy of a single node v_i can be computed in linear time with respect to the number of non-zero entries of p_i.

Recalling that the teleportation parameter α in APPR controls the effective neighborhood size, we detect roles on multiple scales by computing APPR for multiple parameter values $\alpha \in \{\alpha_1, \ldots, \alpha_l\}$ and concatenating for each node its corresponding l entropy values. The final descriptor of node v_i is then given as

$$f_i = \left[H\left(p_i^{(\alpha_1)}\right), \ldots, H\left(p_i^{(\alpha_l)}\right) \right]^T \in \mathbb{R}^l. \tag{11.4}$$

The entire procedure for calculating the role descriptors is sketched in Figure 11.2 and can be summarized as outlined in Algorithm 9. The input for the method is the entire graph dataset $\mathcal{G}$, a list of node labels $\mathscr{L}$, a list of teleportation parameters αs, and the approximation threshold ϵ. For each of the nodes v in $\mathcal{G}$, the algorithm stacks the entropy-based representations of the corresponding APPR vectors, denoted as p_v^α, to generate the role descriptor of v, i.e., emb_v. The algorithm finally fits a classification model on the collection of role descriptors and retrieves the resulting model for node classification.

11.3 Experiments

In the following, we investigate the performance of our new node representations compared to node representations created with different existing techniques including RolX [117] which relies on hand-crafted features, the diffusion-based method GraphWave [82] and the SkipGram-based struc2vec [217] method.

Datasets

To empirically investigate the benefits of our approach for determining the role descriptors, we first use a barbell graph to compare the performance of various structure based node representations. The barbell graph consists of two fully connected components that are connected by a long chain. Specifically, we use a barbell graph that has ten nodes within each clique and a chain of length ten as illustrated in Figure 11.3(a). The goal of this experiment is to demonstrate the performance in terms of identifying different roles.

In a second experiment, we use the mirrored Karate network as already used in [217, 82]. In particular, this dataset consists of two copies of the Zachary's karate network with one edge connecting the two copies by linking one randomly chosen node with its copy. We use this dataset to demonstrate the performance of the embedding techniques in terms of accuracy. Obviously, a node and its copy should end up at the same position (or very close to each other) in the embedding space since they have exactly the same roles.

Finally, we employ two real world datasets, i.e., the European and the American air traffic networks used in [217]. These networks are unweighted and undirected. Nodes represent airports which are connected if there have been commercial flights during the time this dataset was recorded. Both networks have four equally sized classes corresponding to the relative level of activity of the airport. These datasets are used to compare the performance in terms of accuracy as well as to demonstrate the efficiency in terms of computation time.

Structural Node Embedding

We first consider the structural node embeddings and compare the embeddings retrieved by our approach to the representations retrieved by RolX, GraphWave and struc2vec. For all competitors, we used the implementations and recommended configurations as published by the authors. The α parameter which is required to be set for our approach is ranged from 0.1 to 0.9 with a step size of 0.1. For each of the parameters, we get a one dimensional descriptor for every node, referred to as APPRroles$_{\alpha=i}$. We also construct higher dimensional representations by combining the node descriptors for all values of α per node. We refer to this configuration as APPRroles$_{stacked}$. The approximation threshold ϵ for the computations of the APPR vectors is set to $\epsilon = 1E^{-4}$.

Barbell Graph

In Figure 11.3 we provide a visual analysis of how well the APPR based node representations are able to embed the structural properties of the node neighborhoods. Figure 11.3(a) depicts the barbell graph while the remaining plots show 1-dimensional structural node embeddings. For visualization purposes, we project higher dimensional embeddings into 1-dimensional spaces by using PCA to be able to discuss the outcomes in comparison to our 1-dimensional representations. Therefore, we use the node identifiers on the y-axes in our plots to spread the depicted 1-dimensional embeddings along the y-axes such that they do not cover each other. Figures 11.3(b)-11.3(d) show the structural node embeddings for the competitors when projecting them onto the first principal component and normalizing the values (cf. x-axes). The first three images in the lower row depict the results for our approach when using different values for α. Precisely, we show the normalized results for $\alpha \in \{0.9, 0.5, 0.1\}$, which means that we range the exploration of the node neighborhoods from local ($\alpha = 0.9$) to spacious ($\alpha = 0.1$). In Figure 11.3(h), we visualize the node embeddings when stacking the descriptors for all values of α and projecting these embeddings into 1-dimensional space, again using PCA.

The first thing we want to emphasize is that since we do not use any external evaluation measures in this experiment due to lack of ground truth and it is thus difficult to say which of the methods work best on this dataset. This also strongly depends on how to define the roles in this graph. What we can clearly see is that RolX and GraphWave reveal only very few roles, i.e., most nodes of the chain are considered to have the same role. This also applies to struc2vec as we can identify three clusters in the 1-dimensional projection. However, differently to the other methods, struc2vec identifies the node in the center of the chain as an own role. Given our representations, we can see that our approach is much more flexible in terms of role identification. When using a rather large value for α, i.e., defining roles only based on very local neighborhoods, our method also considers the chain elements to have the same role. However, when decreasing the value of α, i.e., enlarging the neighborhoods based on which to define the roles, we can observe that the roles of the nodes in the chain are considered to differ more and more. Nevertheless, the chain nodes that have the same hop distance from the center of the chain are always considered to have the same role. Given that roles are not always defined precisely, this is a very desirable property of our approach that RolX or struc2vec cannot fulfill. GraphWave might be able to have this flexibility but due to operating on the spectra of the graphs rather than in the vertex domain, it might be difficult to set the corresponding parameter appropriately.

Summarizing the insights revealed by this experiment, we can state that the choice of the value for α might have a significant impact on the outcome. In other words, using a small α value leads to an accurate distinction between node roles, while rather large values of α do not distinguish as accurately between different roles.

Mirrored Karate Network

Next, we consider the mirrored Karate network for which we measure the performance of the embedding methods by doing 1-NN-range queries. Recall that this network consists of two copies of the Zachary's Karate network and an additional edge which links a randomly chosen node with its copy. Given a set of structural node embeddings $\mathcal{E}$ that contains one embedding for each node of the mirrored karate network, the goal of this experiment is to identify the copy of the query node among its 1-nearest-neighbors. Note that we call this query 1-NN-range query to emphasize that the set of 1-nearest-neighbors might be of size greater than 1. Precisely, we compute the nearest neighbor o for each of the query points $q \in \mathcal{E}$, and subsequently perform a range query around q with distance $dist(o, q)$.

The results given in Table 11.1 show the accuracy and average size of the queries' candidate sets, i.e., $\varnothing|\mathscr{C}|$. Except for struc2vec, all methods achieve an accuracy of 100%. However, when considering the precision that we measure by the size of the candidate set of the 1-NN-range query, we can see that our approach gives the best result on average.

Airport Traffic Networks

The results for the two airport traffic networks can be reviewed in Table 11.2. As proposed in [217] we employ one-vs-rest classifications with 90-10 train-test splits and report the mean accuracy and standard deviation over 10 runs. Additionally, the table contains the runtimes in milliseconds for each method including the preprocessing steps for struc2vec and our approach. RolX and GraphWave do not have preprocessing steps if executing their standard configurations[1].

For both networks, we can notice that although our proposed method outperforms RolX and GraphWave in terms of accuracy, the scores achieved with struc2vec are slightly better. Note that the results for the European airports network are comparable across all methods. However, regarding the runtime our method clearly outperforms the competitors. Compared to struc2vec our approach is more than 2'300 times faster, even when considering the APPRroles$_{stacked}$ configuration, on the USA airports network[2]. It should be noted that the accuracy of our approach might be further improved by spending more time for preprocessing the local neighborhoods. As we use an approximate version of the Personalized PageRank for engineering the local neighborhoods, it is possible to decrease the approximation threshold to get more accurate node descriptors. However, this comes at the cost of increased computation time.

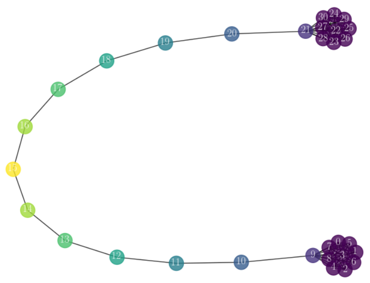
11.4 Conclusion

In this chapter, we presented a novel approach for defining structural node representations that describe the nodes' roles within a graph. More precisely, we figured out that local, personalized PageRank distributions encapsulate the structure of local node neighborhoods and can be compressed to meaningful role descriptors. Compared to previous approaches that tackle the problem of structural node embeddings our approach is fast to compute and allows for much flexibility in terms of identifying different node roles by simultaneously being quite simple to interpret. Our experiments empirically demonstrate the flexibility and show that we can outperform state-of-the-art

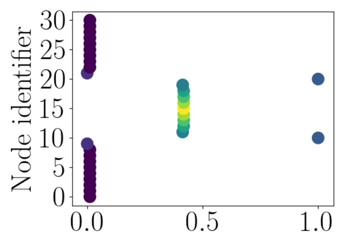
[1]Note that the standard configuration of GraphWave implements an automatic mode for parameter selection. Thus, we must compare its runtime to APPRroles$_{stacked}$.

[2]The EUR Airports network consists of 399 nodes and 5995 edges. The USA Airports network consists of 1190 nodes and 13599 edges.

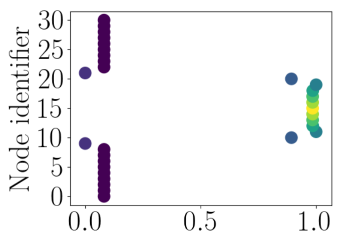
methods when using the role descriptors for node classification tasks. In the next chapter we hence state that these role descriptors are expressive enough to achieve good performance in terms of graph classification when aggregating them for entire graph structures.



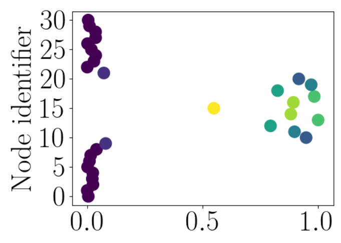
(a) Barbell Graph



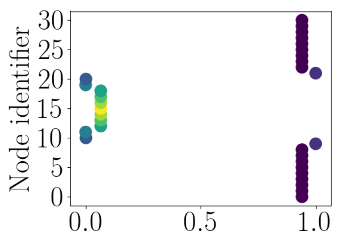
(b) RolX



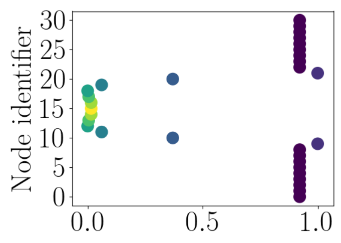
(c) GraphWave



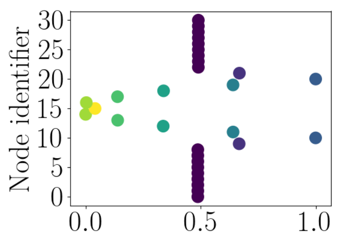
(d) struc2vec



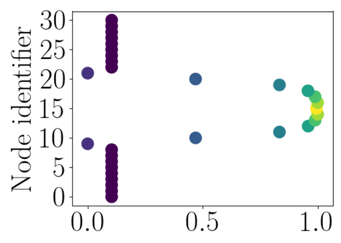
(e) APPRroles$_{\alpha=0.9}$



(f) APPRroles$_{\alpha=0.5}$



(g) APPRroles$_{\alpha=0.1}$



(h) APPRroles$_{stacked}$

Figure 11.3: 1-dimensional node representations for the barbell graph. The y-axes show the node identifiers (i.e., numerical values that we assigned to the nodes), the x-axes are the 1-dimensional node descriptors on [0,1] scale.

Method	Acc.	$\varnothing\lvert\mathscr{C}\rvert$	$\frac{Acc.}{\varnothing\lvert\mathscr{C}\rvert}$
RolX	1.00	6.59	0.151
GraphWave	1.00	6.56	0.152
struc2vec	0.56	3.00	0.186
APPRroles$_{\alpha=0.1}$	0.96	4.53	0.211
APPRroles$_{\alpha=0.2}$	**1.00**	**4.62**	**0.216**
APPRroles$_{\alpha=0.3}$	0.99	5.62	0.175
APPRroles$_{\alpha=0.4}$	1.00	5.00	0.200
APPRroles$_{\alpha=0.5}$	1.00	5.44	0.183
APPRroles$_{\alpha=0.6}$	1.00	4.91	0.203
APPRroles$_{\alpha=0.7}$	0.93	4.35	0.212
APPRroles$_{\alpha=0.8}$	1.00	5.76	0.173
APPRroles$_{\alpha=0.9}$	1.00	6.32	0.158
APPRroles$_{stacked}$	1.00	4.97	0.201

Table 11.1: Results for the 1-NN-range queries on the mirrored Karate network. $\varnothing\lvert\mathscr{C}\rvert$ denotes the average size of the candidate sets retrieved by the 1-NN-range queries.

Method	EUR Airports	t in ms	USA Airports	t in ms
RolX	0.78±0.06	16747	0.77±0.05	66463
GraphWave	0.78±0.05	16596	0.77±0.04	263405
struc2vec	0.81±0.09	695616	0.84±0.05	8300984
APPRroles$_{\alpha=0.1}$	0.77±0.03	375	0.75±0.00	772
APPRroles$_{\alpha=0.2}$	0.76±0.03	337	0.75±0.01	530
APPRroles$_{\alpha=0.3}$	0.75±0.01	209	0.76±0.03	431
APPRroles$_{\alpha=0.4}$	0.75±0.01	198	0.78±0.04	379
APPRroles$_{\alpha=0.5}$	0.77±0.04	189	0.78±0.05	331
APPRroles$_{\alpha=0.6}$	0.77±0.04	153	0.78±0.05	307
APPRroles$_{\alpha=0.7}$	0.76±0.03	143	0.79±0.07	281
APPRroles$_{\alpha=0.8}$	0.74±0.03	134	0.78±0.06	240
APPRroles$_{\alpha=0.9}$	0.75±0.00	50	0.78±0.06	252
APPRroles$_{stacked}$	**0.79±0.07**	**1788**	**0.80±0.07**	**3523**

Table 11.2: Results for one-vs-rest classification. We report the mean accuracy and standard deviation for each configuration. The table also reports runtime measurements for creating the representations including preprocessing steps.

Chapter 12

Unsupervised Graph Embedding

The work presented in this chapter has been published as the article *Structural Graph Representations based on Multiscale Local Network Topologies* in the Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence (WI), 2019 [46].

12.1 Introduction

Many applications benefit from using representations of entire graph structures for downstream machine learning tasks. Specific application scenarios include target group analysis, e.g., in marketing where one might aim at identifying similar interest groups within social networks to exploit new target groups, or pharmaceutical applications, e.g., drug design where small molecule structures may activate or inhibit the functions of specific proteins. However, for being able to use network structures for downstream clustering or classification tasks, it is important to get suitable vector representation of that networks. Although it might also be possible to define specific distance measures like the graph edit distance for downstream tasks, these measures have the disadvantage that they tend to be computationally expensive (especially when dealing with many graph structures), and oftentimes do not fit the problem at hand. Thus, it is often more practicable to learn representations.

In this chapter, we tackle the problem of learning representations for entire graph structures in an unsupervised manner. Precisely, we argue that graphs can generally be described by the composition of their nodes' functions. The inductive bias of our approach can be justified by taking the following, simplified example into account. Consider a set of social community circles extracted from a virtual social network that is composed of two kinds of communities, i.e., soccer teams and Youtube circles, and a vendor for sports products who wants to start an advertising campaign. This vendor is obviously interested in advertising his products in the soccer team communities. In fact, the functions, or roles, of the single members of the communities can be used to distinguish the communities into the two classes. While the roles of the members of the soccer team communities are expected to be the same across each of the corresponding graphs (because soccer team networks are likely to be clique-like), the roles in the Youtube circles are probably not evenly distributed (there is typically one user who acts as the channel host and forms the center of a star-like pattern). Thus, we claim that the nodes' roles are indeed useful to represent entire graph structures. In the following, we therefore re-use the role descriptors described in the previous chapter and use them for the task of classifying entire graphs. Formally, for graph classification, we are given a label vector $y \in \{1, \dots, K\}^N$ assigning each graph $G_i \in \mathcal{G}$ to one of K classes. The goal is to learn a model, which accurately predicts the class label y_i given G_i. By using an unsupervised clustering algorithm on local, i.e., for single graphs, and global, i.e., for all graphs in the network, scales, we discretize the definition of roles and represent each graph by a count vector for role types.

Our experimental evaluation shows that even simple aggregation schemes already lead to results that are comparable to the ones retrieved by current state-of-the-art methods.

12.2 Aggregated Graph Descriptors

Since our structural node descriptors are location-invariant and thus transferable among different graphs, we are able to compare whole graphs by comparing their respective sets of node descriptors. However, simply collecting the node descriptors in a (ordered) set for each graph is not a straightforward solution, since this would result in different length representations for graphs with different numbers of nodes. To this end, we propose two histogram-based aggregation schemes to discretize the notion of roles. Precisely, the first aggregation scheme is based on classic histograms with all bins having the same size, while the second aggregation scheme can be interpreted as an adaptive histogram where the bins adapt to the distribution of role descriptors. For both aggregation schemes, the node roles can be aggregated in a local or global fashion, i.e., a graph representation for a single graph either relies solely on the node roles that appear in that graph, or the representation relies on global role notions that are defined over all graphs. The intuition behind defining roles on a local view is to be more robust against outlier roles. Recalling that role descriptors are continuous values, outliers may affect the value range over all role descriptors on a global view such that some equi-sized histogram bins defined over the entire value range become meaningless. On the other hand, the local view approach has the underlying assumption that role notions do not differ substantially over the entire body of graphs.

Somewhat more formally, let $\mathcal{G} = \{G_1, \ldots, G_N\}$ be a set of graphs, with $G_i = (V_i, E_i)$ being a graph, $V = \bigcup_{i=1}^{N} V_i$ denoting the set of vertices and $E = \bigcup_{i=1}^{N} E_i$ being the set of edges. Furthermore, let $f(v_j) \in \mathbb{R}^d$ denote the continuous role descriptors of nodes $v_j \in V$ in $\mathcal{G}$, as described in the previous chapter. To derive graph embeddings that follow the baseline aggregation scheme, i.e., the aggregation scheme that uses equi-sized bins for the histogram representations, we simply aggregate the nodes of graph by discretizing the value range of the role descriptors into equi-sized bins and counting the occurrences of role descriptors per graph and bin. Hence, the graph representation F_i for graph G_i is defined as

$$F_i = [|\{v \in V_i \mid b_j \leq f(v) < b_{j+1}\}| : j = 1, \ldots, k]^T \in \mathbb{R}^k, \tag{12.1}$$

with b_j being the lower bound value of the j-th histogram bin, and k denoting the number of bins. The definition of the bins' value ranges allows to generate graph representations from a global or a local perspective. To define representations on a global view, the set of bins $\mathcal{B}$, with $|\mathcal{B}| = k$, is defined over the value range of role descriptors collected from all graphs in the training dataset $G_{train} = \{G_0, \ldots, G_{n-1}\} \subseteq \mathcal{G}$, i.e.,

$$\begin{aligned} \mathcal{B} &= \bigcup_{j=0}^{k-1} [b_j, b_{j+1})\,, \text{with} && (12.2) \\ b_j &= \begin{cases} min(\{f(v) \mid v \in \bigcup_{i=0}^{n-1} V_i\}) & if\ j = 0, \\ \frac{max(\{f(v)|v \in \bigcup_{i=0}^{n-1} V_i\}) - min(\{f(v)|v \in \bigcup_{i=0}^{n-1} V_i\})}{k} & else. \end{cases} && (12.3) \end{aligned}$$

In contrast, to define graph representations on a local view, the sets of bins $\mathcal{B}_i$, with $|\mathcal{B}_i| = k$ and $0 \leq i < n$, are defined for each graph $G_i \in G_{train}$ individually, i.e.,

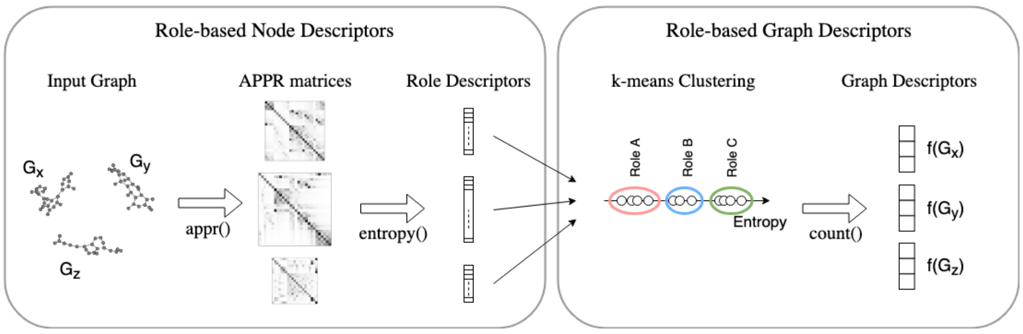


Figure 12.1: Workflow for calculating the role-based graph descriptors.

$$\mathcal{B}_i = \bigcup_{j=0}^{k-1} [b_j, b_{j+1})\,, \text{with} \tag{12.4}$$

$$b_j = \begin{cases} min(\{f(v) \mid v \in V_i\}) & if\ j = 0, \\ \frac{max(\{f(v)|v \in V_i\}) - min(\{f(v)|v \in V_i\})}{k} & else. \end{cases} \tag{12.5}$$

Similarly, we derive graph embeddings that rely on the aggregation scheme that is based on adaptive histograms as follows. First we collect node descriptors from all graphs (in the global setting) or for each graph individually (in the local setting) in the training dataset and cluster them with k-Means [174]. The resulting cluster centers $\{\mu_i \in \mathbb{R}^l \mid i = 1, \dots, k\}$ can be interpreted as multi-scale role concepts appearing in the dataset. In a second step, we assign each node v in a given graph G_i to its nearest cluster center $\mu(v)$ and use the resulting count vector

$$F_i = [|\{v \in V_i \mid \mu(v) = \mu_j\}| : j = 1, \dots, k]^T \in \mathbb{R}^k, \tag{12.6}$$

as representation for that graph. One important advantage of these graph descriptors in general is that they can be computed very efficiently, i.e., in linear time with respect to the total number of nodes in the dataset. Furthermore, the number of clusters k can be varied flexibly to explore different numbers of roles in a graph. For a supervised objective, the hyper-parameter can simply be optimized over a range of sensible values. However, other clustering techniques may be employed for discretizing the continuous role descriptors, too.

Figure 12.1 extends the workflow presented in the previous chapter by additionally calculating the described graph descriptors using the global, adaptive approach. The final procedure consists of two blocks: in the first block,

Dataset	$\lvert\mathcal{G}\rvert$	$\lvert\mathscr{L}\rvert$	$\phi\lvert V\rvert$	$\phi\lvert E\rvert$
MUTAG	188	2	17.93	19.79
ENZYMES	600	6	32.63	62.14
NCI1	4110	2	29.87	32.30
NCI109	4127	2	29.68	32.13
PROTEINS	1113	2	39.06	72.82
IMDB-BINARY	2000	2	429.63	497.75
IMDB-MULTI	1500	3	13.00	65.94
REDDIT-BINARY	2000	2	429.63	497.75
REDDIT-12K	11929	11	391.41	456.89
REDDIT-5K	4999	5	508.52	594.87

Table 12.1: Benchmark datasets for graph classification. The upper part of the table contains biological networks, the lower part of the table refers to social network datasets. $|\mathcal{G}|$ denotes the number of graphs, $|\mathscr{L}|$ is the number of classes and $\phi|V|$, resp. $\phi|E|$ is the average number of nodes, resp. edges.

the continuous role descriptors for each node are calculated. Given the raw network – composed of multiple, differently sized components which form graph structures on their own – as input, we compute the stationary APPR distributions for each node. From these, we next derive the continuous role-based node descriptors by computing the entropy values of the distributions for each node. Stacking these entropy values for each component results in differently sized and thus incomparable vectors (or matrices in case of multiple α values for the calculations of the APPR distributions). In order to enable comparisons between differently sized subgraphs, we first discretize the notion of roles by employing the k-means algorithm on the continuous role descriptors in the second block of our procedure[1]. Secondly, for each of the subgraph structures, we count the appearances of each role within the corresponding network to construct equally-sized graph descriptors which can easily be used for downstream tasks like classifications. Note that the example depicts the procedure for a single value of α used for APPR. As we show in the experiments section, richer representations can be calculated by using multiple values for α.

12.3 Experiments

We evaluate the proposed graph representations by employing several graph classification tasks. We compare our approach with various settings against state-of-the-art approaches including Deep Graph Kernels (DGK) [269], the diffusion-based NetLSD [252] method and NetSimile [34] which relies on hand-crafted features. The settings used for our approach include the local and global setups for both aggregation schemes, the baseline scheme that relies on classic histograms and the adaptive approach that relies on adaptive histograms. For the latter, we also experiment with two setups for the calculation of the role representations. The first setup computes role descriptors from a single neighborhood that is captured by employing the approximate Personalized PageRank with $\alpha = 0.1$, and the second setup concatenates role descriptors gathered from multiple, differently scaled neighborhoods for whose computation we ranged the α value from 0.1 to 0.9 with a step size of 0.1.

Datasets

For evaluating the performance of the proposed graph representations in terms of graph classification, we use several biological and social networks which are taken from [141] and are summarized in Table 12.1.

The MUTAG dataset consists of graph structures that represent chemical compounds which are classified into two classes depending on their mutagenic effect on a bacterium. The two NCI datasets also consist of chemical compounds and the class labels correspond to their activity against cancer cells. The protein graphs in the PROTEINS dataset are either enzymes or non-enzymes. The ENZYMES dataset contains protein graph structures that are categorized into 6 different top-level enzyme classes.

The IMDB graph datasets consist of ego networks where nodes are actors or actresses and edges connect actors/actresses if they have been in the same cast. The labels of the graphs (two distinct labels for IMDB-BINARY and three for IMDB-MULTI) correspond to the genre of the corresponding movie. For all three REDDIT datasets, nodes correspond to users and links indicate that one user responded to the comment of the other user. The classes of the REDDIT-BINARY dataset indicate that a given graph either stems from a question/answer-based community or a discussion-based community. The classes of the REDDIT-5K and REDDIT-12K graphs correspond to subreddits, with REDDIT-5K containing discussion graphs from 5

[1]In the figure, we used $k = 3$ as the number of roles.

distinct subreddits and REDDIT-12K consisting of graphs from 11 different subreddits.

Results

After presenting the results of our experimental study for structural node representations in the previous chapter, we show that the proposed node representations are well-suited for graph classification when aggregating them for entire graph structures. Therefore, we compare our graph representations against Deep Graph Kernels (DGK), NetLSD and NetSimile. Though the authors of DGK present three kernel types, i.e., graphlet kernels, shortest path kernels and Weisfeiler-Lehmann kernels, note that the latter two require additional node labels which are considered by none of the other methods. For the sake of a fair comparison, we only compare against graphlet kernels. For NetLSD, we use both presented configurations, i.e., the variant which uses heat trace signatures, denoted as NetLSD$_{Heat}$, as well as the variant that uses wave trace signatures, denoted as NetLSD$_{Wave}$. Either way, we calculate the full eigenspectra rather than using one of the proposed approximations. Regarding our approach, we present the results for $\alpha = 0.1$ and the variant where we stack the representations for all values of α.

Method	MUTAG	ENZYMES	NCI1	NCI109	PROTEINS
DGK	0.827±0.021	0.271±0.008	0.625±0.003	0.627±0.002	0.717±0.005
NetLSD$_{Heat}$	0.841±0.032	0.367±0.042	0.665±0.014	0.647±0.018	0.649±0.023
NetLSD$_{Wave}$	0.809±0.055	0.265±0.042	0.611±0.012	0.603±0.016	0.603±0.028
NetSimile	0.833±0.034	0.387±0.046	0.676±0.019	0.663±0.018	0.602±0.023
PPRroles$^{global}_{\alpha=0.1}$	**0.871±0.043**	0.340±0.033	0.704±0.019	0.699±0.014	**0.665±0.042**
PPRroles$^{global}_{stacked}$	0.858±0.043	**0.398±0.046**	**0.732±0.012**	**0.734±0.013**	0.644±0.028
PPRroles$^{local}_{\alpha=0.1}$	0.826±0.052	0.263 ±0.041	0.610±0.011	0.607±0.020	0.636±0.034
PPRroles$^{local}_{stacked}$	0.827±0.065	0.247 ±0.038	0.626±0.014	0.615 ±0.015	0.628 ±0.020
PPRroles$^{global}_{histogram}$	0.841 ±0.039	0.332 ±0.038	0.697 ±0.014	0.691 ±0.012	0.637 ±0.044
PPRroles$^{local}_{histogram}$	0.895 ±0.034	0.266 ±0.049	0.657 ±0.015	0.648 ±0.014	0.607 ±0.028

Table 12.2: Accuracy in 1-NN classification on biological networks. Note that the DGK results are taken from [269] and report the achieved classification accuracy when applying a SVM.

For all experiments in this section, we report the mean accuracy of 1-NN classifications over 10 runs. The results for DGK are borrowed from the original paper since the method relies on a Kernel-SVM with precomputed kernels and the authors used the same evaluation setup.

Method	IMDB-BINARY	IMDB-MULTI	REDDIT-BINARY	REDDIT-12K	REDDIT-5K
DGK	0.670±0.006	0.446±0.005	0.780±0.004	0.322±0.001	0.413±0.002
NetLSD$_{Heat}$	0.690±0.034	0.421±0.035	0.782±0.017	0.246±0.009	0.332±0.014
NetLSD$_{Wave}$	0.681±0.033	0.420±0.038	0.671±0.023	0.227±0.005	0.317±0.012
NetSimile	**0.704±0.039**	0.415±0.039	0.848±0.010	0.335±0.009	0.413±0.012
PPRroles$_{\alpha=0.1}$	0.645±0.032	0.404±0.041	0.857±0.017	0.348±0.009	**0.427±0.014**
PPRroles$_{stacked}$	0.666±0.035	**0.429±0.039**	**0.867±0.014**	**0.348±0.007**	0.417±0.010
PPRroles$^{local}_{\alpha=0.1}$	0.663±0.028	0.419±0.041	0.772±0.016	0.277±0.008	0.373±0.015
PPRroles$^{local}_{stacked}$	0.642±0.042	0.405±0.025	0.774±0.015	0.274±0.008	0.376±0.010
PPRroles$^{global}_{histogram}$	0.649±0.033	0.398±0.039	0.839±0.015	0.341±0.007	0.426±0.014
PPRroles$^{local}_{histogram}$	0.638±0.028	0.406±0.034	0.815±0.015	0.309±0.005	0.418±0.014

Table 12.3: Accuracy in 1-NN classification on social networks. Note that the DGK results are taken from [269] and report the achieved classification accuracy when applying a SVM.

Biological Networks

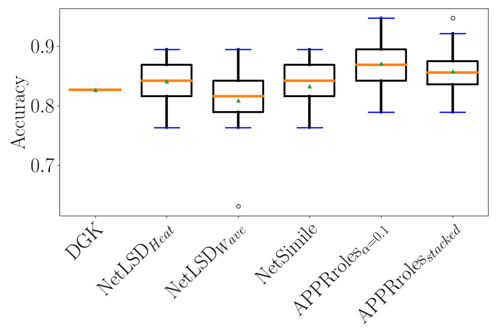
The results for the biological datasets are listed in Table 12.2. On the first four datasets our approach achieves better accuracy scores than the competitors. Indeed our approach reaches gains of approx. 3% on MUTAG, 2.8% on ENZYMES, about 8.2% on NCI1 and over 10% on NCI109 compared to the best competitor. For PROTEIour classifier, our method achieves the best score. In Figure 12.2, we report the results of 20 runs per method in somewhat more detail for the biologNS, the deep graph kernel method shows the best accuracy. However, among the methods that we evaluate with ical network datasets. Note that for the sake of readability we only report the best settings of our approach, i.e., the global approach using adaptive histograms.

Social Networks

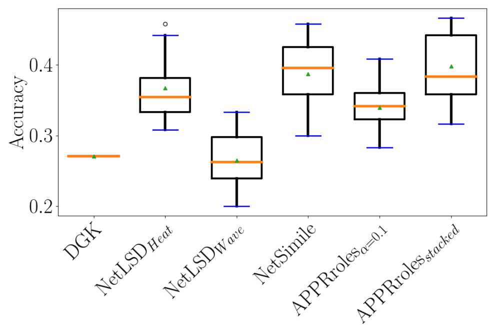
Finally, considering the social networks, we can observe similar results. As can be seen in Table 12.3, except for IMDB-BINARY, our method outperforms all competitors for which we applied the 1-NN classification. Though we want to note that the deep graphlet kernel was slightly better on IMDB-MULTI when using the evaluation method proposed in [269]. In Figure 12.3, we again report more detailed results of 20 runs per method for the social network datasets. Again, we only report the results of the global approach using adaptive histograms to ease the readability.

12.4 Conclusion

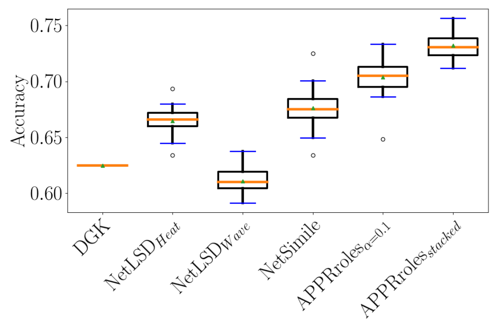
In this chapter, we employed the role descriptors presented in the previous chapter to generate embeddings for entire graph structures. We proposed a straightforward aggregation scheme that uses traditional histograms with equi-sized bins to aggregate the role descriptors for each graph. Further, we used a somewhat more sophisticated aggregation scheme that also relies on histograms but is adaptive to the distribution of all nodes' role descriptors due to being based on the k-means clustering. Our experimental evaluation showed that even with these simple aggregation schemes, we were already able to outperform state-of-the-art techniques. Hence, we conclude that the nodes' functions, resp. roles, and in particular their role descriptors (and not only the ones presented in the previous chapter) are indeed useful when aiming at representing entire graph structures as numerical vectors.



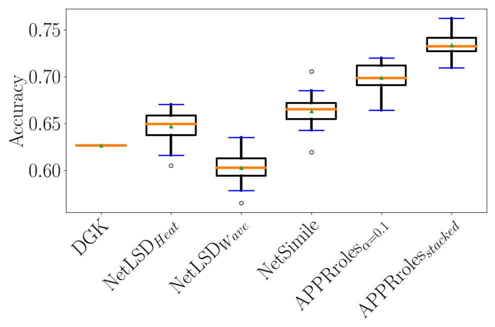
(a) Accuracy for MUTAG.



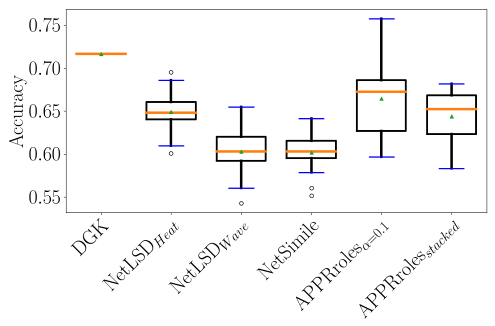
(b) Accuracy for ENZYMES.



(c) Accuracy for NCI1.

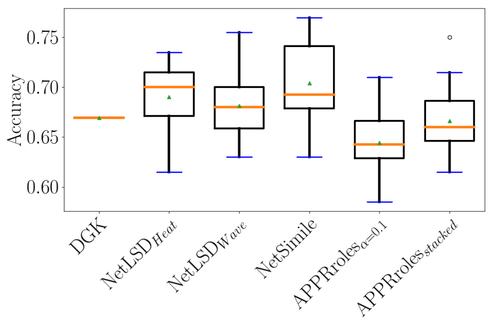


(d) Accuracy for NCI109.

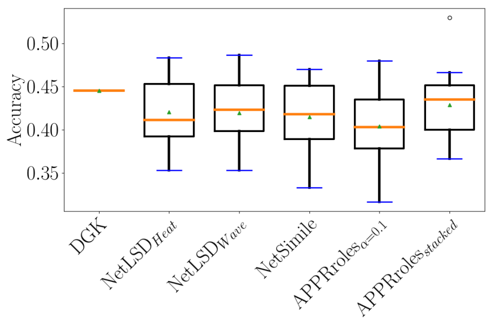


(e) Accuracy for PROTEINS.

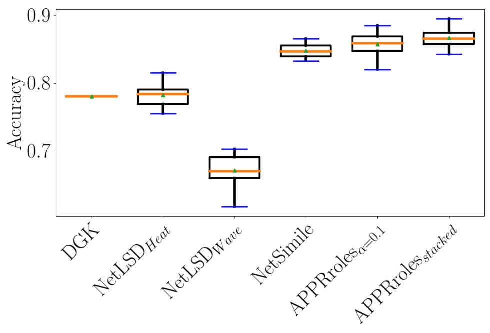
Figure 12.2: Accuracy scores achieved with 1-NN classification on the biological datasets. The orange lines denote the medians, the green triangles depict the mean values and correspond to the values reported in Table 12.2. Again, the DGK results are taken from [41] and report the achieved classification accuracy when applying a SVM.



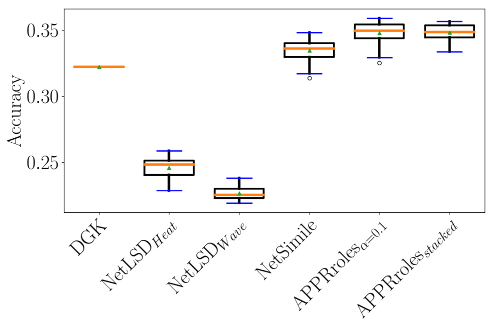
(a) Accuracy for IMDB-BINARY.



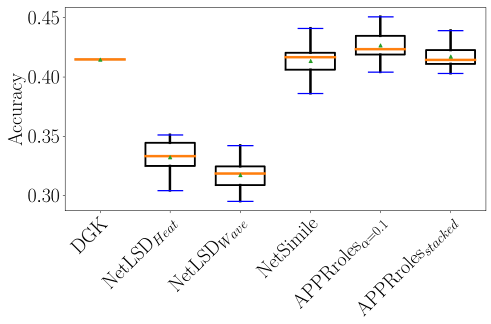
(b) Accuracy for IMDB-MULTI.



(c) Accuracy for REDDIT-BINARY.



(d) Accuracy for REDDIT-12K.



(e) Accuracy for REDDIT-5K.

Figure 12.3: Accuracy scores achieved with 1-NN classification on the social datasets. The orange lines denote the medians, the green triangles depict the mean values and correspond to the values reported in Table 12.3. Again, the DGK results are taken from [41] and report the achieved classification accuracy when applying a SVM.

Chapter 13

Semi-Supervised Learning on Graphs

The work presented in this chapter has partly been published as the article *Semi-Supervised Learning on Graphs Based on Local Label Distributions* on the 14th ACM KDD International Workshop on Mining and Learning with Graphs, 2018 [90]. A preliminary version can be found on arXiv [89]. At date, a full research paper version is under review for the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases, 2020.

13.1 Introduction

The increasing relevance of graph-structured data has been accompanied by an increased interest in learning algorithms which can leverage underlying graph structure to make accurate predictions about the modeled entities. In many of these applications, it is important to categorize data objects into several types such as user classes, functional types or content topics. In many cases, we have these types only available for a portion of the nodes in the network. An important task is now to predict the types or more generally speaking the *labels* of the nodes where they are still unknown.

Real-world data is often complex and in order to make accurate predictions, different aspects need to be taken into account. Strong assumptions, such that a node has the same label as most of its neighbors or that correlations between differently labeled nodes are known a priori, may lead to insufficient exploitation of more complex correlations present in the data. *Homophily*-based approaches [285, 263, 214, 103, 244, 260] assume that nodes which are closely connected in the graph, should have similar labels.

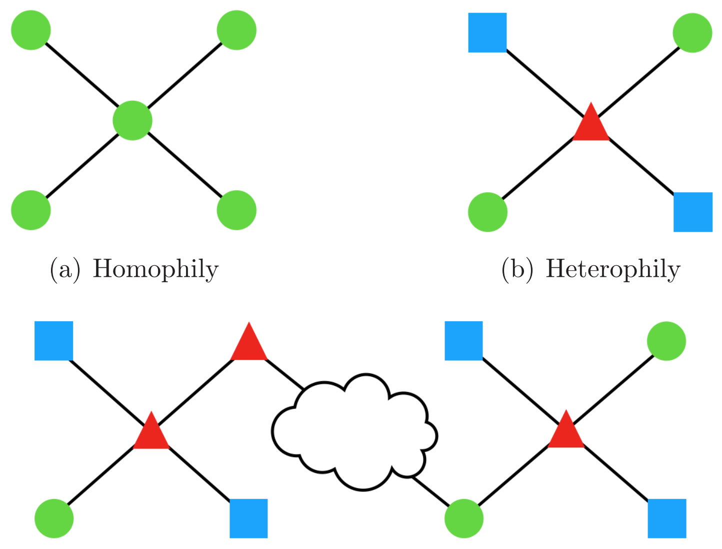


Figure 13.1: Different node classification methods rely on different assumptions on node similarity, e.g., neighboring nodes have similar labels (13.1(a), Homophily) or exhibit specific correlations between labels (13.1(b), Heterophily). Our method adaptively learns different types of correlations (13.1(a), 13.1(b) and 13.1(c)) appearing (possibly simultaneously) in the same graph for different labels. It also detects patterns at multiple scales and uses information from the whole graph (also from different connected components).

This assumption holds, e.g., in graphs where an edge denotes similarity between two instances. However, relationships between two entities modeled by edges in a graph generally may describe any interactions between them rather than being restricted to model only similarity. Another shortcoming of homophily-based methods is that they cannot use information from distant parts of the graph, different connected components or from other graphs with similar structure and labels for the classification decision. These shortcomings are shared by existing approaches which support *heterophily* in graphs [206, 144, 97, 208]. Heterophily is the tendency to connect to nodes with different labels. An example is an heterosexual dating network with the gender as a label. Additionally, existing methods either assume that all labels follow the same homophilic or heterophilic pattern, or correlations between pairs of labels must be provided explicitly. However, in many scenarios it is not feasible to manually model correlations between labels. At the same time, many real-life graphs are characterized not only by some specific mix of

Method	Homophily	Heterophily	Mixed	Local Variation	Adaptive	Labels	Remote
Homophilic Node Embedding [214, 103, 244, 62, 260, 91, 43, 183, 279]	✓	✗	✗	✗	✗	✗	✗
Label Diffusion [284, 285, 281, 208, 259]	✓	✓	✗	✗	✗	✓	✗
Belief Propagation [206, 144, 97]	✓	✓	✓	✗	✗	✓	✗
Planetoid [271]	✓	✗	✗	✗	✗	(✓)	(✓)
MPNN [98]	(✓)	(✓)	(✓)	(✓)	(✓)	✗	✓
Ada-LLD	✓	✓	✓	✓	✓	✓	✓

Table 13.1: Comparison to related node classification methods based on whether they fulfill (✓) the desired key properties or not (✗). Parentheses indicate partial fulfillment.

heterophily and homophily, but rather show high variations of these patterns for the same labels across the same graph [209]. Another problem of current methods is that the proximity of the considered neighbors often is not taken into account explicitly. However, knowing how far a particular neighbor is may be beneficial. E.g., in real-world social graphs the friends of some friends may contain more information about a node's label than the direct friends themselves [21].

In this chapter, we propose a label-based approach *Ada-LLD* which *learns* all possible correlations of a node's label with labels in its local neighborhood, as depicted in Figure 13.1, at multiple scales. The main idea is as follows: If the graph structure is useful for prediction, then there should exist a correlation between the label of a node and the distribution of labels in its local neighborhood. Our approach neither requires this correlation to be predefined in advance nor to be the fixed across the graph. Therefore, our method first determines the distributions of labels in neighborhoods of different extensions. To do so, we consider only the most relevant neighbors for each node, which we determine by using *Approximate Personalized PageRank*. Given the representations of label distributions from various parts of the graph, our method learns how to infer labels for unlabeled nodes based on these representations. An important advantage of our approach is that it is able to use information from the whole graph (or all available graphs) for classifying a node and that it does not make any assumptions about relationships of labels (it learns them instead).

We summarize our main contributions as follows:

- A novel approach to semi-supervised node classification which is able to learn different types of correlations between labels by considering local label distributions.

- Variations of our base model for detecting and combining label correlations at multiple scales.

- An efficient and scalable algorithm for computing local label distributions.
- Thorough experimental evaluation of our models and comparison with state-of-the art methods on several real world datasets.

13.2 Related Work

As discussed in Chapter 9.3, numerous approaches related to semi-supervised node classification have been proposed recently. These can broadly be categorized into unsupervised node embedding techniques and semi-supervised techniques. To position the work presented in this chapter within the state-of-the-art, we compare related methods for node classification with respect to several key properties in the following.

Comparison of related methods

We compare those related methods with respect to key properties which describe different aspects of how nodes can be related with each other for classification. In addition to the discussion below, an overview is provided in Table 13.1.

- **Homophily:** Whether the method is able to model homophily. Label Diffusion, MPNN and Ada-LLD are able to learn homophily, but are not doing so explixitly. MPNN however is only able to model homophily indirectly via additional node attributes. The remaining methods can either be parametrized explicitly to model homophily or focus on homophily by design.
- **Heterophily:** Whether the method is able to model heterophily. MPNN and Ada-LLD are able to model heterophily, but are not doing so explicitly. Again, MPNN is only able to model heterophily indirectly via additional node attributes. While Belief Propagation needs to be parametrized explicitly to model heterophily, advanced Label Diffusion methods focus on heterophily by design.
- **Mixed:** Whether the method is able to model homophily or different heterophily patterns for different labels at the same time. Capabilities of related methods w.r.t. this property are analogous to the heterophily property above.

- **Local Variation:** Whether the method is able to model homophily or different heterophily patterns for the same label at the same time. The only methods capable of modeling this property are MPNN and Ada-LLD , where MPNN again relies on additional node attributes.
- **Adaptive:** Whether the model is able to adaptively learn appropriate node similarities without explicitly relying on either homophily or heterophily. MPNN learns general correlations only between a node's label and additional node attributes appearing in the neighborhood. Ada-LLD learns such correlations directly from the neighboring node labels. The remaining methods are not adaptive.
- **Labels:** Whether node similarity is modeled directly based on labels. Homophilic graph embeddings are unsupervised and only consider graph topology. MPNN models node similarity based on node attributes. Planetoid considers labels directly only via sampling of nodes with the same labels. Label Diffusion and Belief Propagation consider labels directly by diffusing them through the graph. Ada-LLD is the only method which directly learns general correlations between a node's label and neighboring labels.
- **Remote:** Whether information from different parts of the graph can be incorporated to classify a node. MPNN and Ada-LLD are location-invariant and thus able to learn correlations based on the whole graph. Planetoid samples nodes with same class labels from different parts of the graph but only in addition to neighboring nodes. The remaining methods only take close-by nodes into account.

In conclusion, none of the existing methods fulfills all of the desired properties. Ada-LLD is the first method to support adaptive learning of direct label-based node similarity for node classification without a need of additional attributes.

13.3 Adaptive Node Similarity Using Local Label Distributions

Problem Setting

A graph $G = (V, E)$ can be represented by an $n \times n$ adjacency matrix $A = (a_{ij})_{v_i, v_j \in V}$, where $a_{ij} \in \mathbb{R}$ denotes the weight of the edge (v_i, v_j). In case of an unweighted graph, $a_{i,j} = 1$ indicates the existence and $a_{i,j} = 0$ the absence of an edge between v_i and v_j. Furthermore, we do not allow self-links and we do not consider further attributes in addition to class labels.

Our problem setting is *semi-supervised node classification*, where the node set V is partitioned into a set of labeled nodes L and unlabeled nodes U, such that $V = L \cup U$ and $L \cap U = \emptyset$. Thereby, each node $v_i \in V$ is associated with a label vector $y_i \in \{0,1\}^l$, where l is the number of possible labels and an entry one indicates the presence of the corresponding label for a certain node. The available labels can be represented by an $n \times l$ label matrix Y_{train}, where the i-ths row of Y_{train} corresponds to the label vector y_i of v_i if $v_i \in L$. For unlabeled nodes, we assign constant zero vectors. The task is to train a classifier using A and Y_{train} which accurately predicts y_i for each $v_i \in U$. In *multi-class* classification, each node is assigned to exactly one class. *Multi-label* classification denotes the general case, in which each node may be assigned to one or more classes and the goal is to predict all labels assigned to a particular node.

The Ada-LLD Model

The main idea of our approach is to learn general correlations between a node's label and the labels of neighboring nodes. According to this intuition, our core model predicts the label vector y_i for a given node v_i as

$$y_i = f\left(aggr\left(\{y_j \mid v_j \in \mathcal{N}(v_i)\}\right)\right), \tag{13.1}$$

where $\mathcal{N}(v_i)$ denotes the neighborhood of v_i, $aggr$ is an aggregation function and f is a classifier which predicts node labels based on the aggregated neighboring node labels.

A sensible choice for $aggr$ would be a weighted function which does not treat any labels equally but assigns more importance to labels of nodes which are more important to v_i. Additionally, a probabilistic interpretation can be obtained by using probabilistic weights. These considerations lead to *local label distributions* which will be introduced in Section 13.3 and are used as input features to our model.

For the classifier f, our default choice is a neural network with a single hidden layer H_1 for which different alternatives will be introduced in Section 13.3. For prediction we use a fully connected layer

$$H_2 = q\left(W_{out} H_1 + b_{out}\right), \tag{13.2}$$

where $W_{out} \in \mathbb{R}^{h \times l}$ is the weight matrix, $b_{out} \in \mathbb{R}^l$ denotes the bias and q is the *softmax* activation $P(c_i) = {\exp(h_i)}/{\sum_{j=0}^{l} \exp(h_j)}$ in case of multi-class classification. If the classification problem is a multi-label one, class probabilities are computed using the *sigmoid* activation $P(c_i) = {}^1/_{(1+\exp(-h_i))}$. The resulting model is quite simple and efficient, yet sufficiently expressive to provide accurate predictions. Details regarding training will follow in Section 13.3.

Local Label Distributions

In many real-world graphs, most of the nodes can be reached within a few steps and often only a small set of neighboring nodes are important to a particular node. Therefore, it is crucial to take only the labels of the most relevant neighbors into account and to weight them accordingly. Considering simple neighborhoods such as the k-hop neighborhood may lead to considering irrelevant nodes or missing important ones. Nodes with a larger shortest path distance could still be more important, e.g., if they can be reached via many different short paths. For similar reasons, nodes with a small shortest path distance might not be equally important. Such effects are captured by *Personalized PageRank (PPR)*. Personalized PageRank can be viewed as a special case of the *PageRank* algorithm [201], where the probabilities in the starting vector are biased towards some set of nodes. We consider the special case in which the starting vector is a unit vector, resulting in personalized importance scores for the particular source node. The PPR-vectors of all nodes in the graph can be stored as rows of a sparse PPR-matrix $\Pi \in \mathbb{R}^{n \times n}$.

Local push-based algorithms [133, 33] can be used to compute *Approximate Personalized PageRank (APPR)* very efficiently and lead to sparse solutions where small, irrelevant entries are omitted. In particular, we consider the algorithm proposed in [234] as outlined in Algorithm 10. The algorithm requires two parameters to be set by the user. The teleportation parameter α determines the effective size of the neighborhood considered for the source node. The second parameter ϵ is a threshold which controls approximation quality and runtime.

Given the PPR-vector π_i and the label matrix Y_{train}, we aggregate the neighboring labels of v_i as follows:

$$ld_i = \pi_i Y_{train} \in R^l \tag{13.3}$$

We call the resulting vector the *label distribution vector* of v_i. Intuitively, the entry $ld_{i,j}$ corresponds to the probability that a random walk starting at v_i stops at a node with label c_j.

Multi-Scale Neighborhood Combination

Label distributions can be expected to change with increasing neighborhood size and an optimal scale will depend on the graph and even on the particular

Algorithm 10 *Compute_LD(v, α, ϵ)*

Input: Source node v, Teleportation probability α, Approximation threshold ϵ, Label matrix Y_{train}

Output: Label distribution vector ld

```
1: // Compute APPR-vector for node $v$
2: $p = \vec{0}$, $r = \vec{0}$
3: $r(v) = 1$
4: while $r(u) \geq \epsilon d(u)$ for some vertex $u$ do
5:    pick any $u$ where $r(u) \geq \epsilon d(u)$
6:    $p(u) = p(u) + (2\alpha/(1+\alpha))r(u)$
7:    for $v$ with $(u, v) \in E$ do
8:       $r(v) = r(v) + ((1-\alpha)/(1+\alpha))r(u)/d(u)$
9:    end for
10:   $r(u) = 0$
11: end while
12: // Compute label distribution vector $ld$ for node $v$
13: $p(v) = 0$
14: $ld = pY_{train}$
15: return $ld$
```

source node itself. Further, information from multiple scales may be combined to detect multi-scale patterns. Performing a parameter search over α would not only be expensive but also lead to a single global scale applied to all nodes in the graph. Instead, we propose to consider label distributions at a small set of different scales and suggest several possibilities of combining them.

We start with computing local label distributions for a set of different teleportation parameters $\{\alpha_1, \dots, \alpha_k\}$, resulting in a tensor

$$X = [X_{\alpha_1}, \cdots, X_{\alpha_k}] \in \mathbb{R}^{k \times n \times l}, \tag{13.4}$$

where slice $X_\alpha = \Pi Y_{train} = (ld_{i,j}) \in \mathbb{R}^{n \times l}$ contains the label distributions w.r.t. α of all nodes in the graph as rows. If only a single fixed scale α is used, the matrix $X = X_\alpha$ is fed to a fully-connected hidden layer H_1 which is then connected to the prediction layer H_2. For the general multi-scale case, we propose four different neural network architectures with different inductive bias for the task of learning the importance of different scales and combining the provided label distributions. Our base model uses only a single hidden layer H_1. Naturally, the model could be extended by an arbitrary number of additional fully-connected hidden layers depending on the task at hand. Since our goal is to prove our concept and we want to keep our architecture comparable with the competing methods, we will restrict ourselves in the following to models with only these combining layers as the single hidden layers.

Average. Our first model, *LD_ AVG*, tries to determine an optimal combination of label distributions in different neighborhood extensions simultaneously for the whole graph. Therefore, we propose to combine the label distributions by taking the weighted average. The weight γ_i for the i-th matrix X_{α_i} is a scalar and is trained jointly with the remaining model parameters. Formally, the hidden layer is computed as

$$H_1 = q\left(\left(\gamma_1 X_{\alpha_1} + \cdots + \gamma_k X_{\alpha_k}\right) W_{avg} + b_{avg}\right), \tag{13.5}$$

with $W_{avg} \in \mathbb{R}^{l \times h}$, $b_{avg} \in \mathbb{R}^h$ and h denoting the number of hidden neurons in the first layer. As default activation q we choose the *ReLU* function (also for the following models). Though this method has a moderate number of parameters, it is not able to learn different scale combinations.

Concatenation. With our second model, which we refer to as *LD_ CONCAT*, we attempt to minimize the assumptions on the data by allowing arbitrary relations between scales. Thus, we concatenate the neighborhood matrices X_{α_i} and learn a representation by applying a fully connected layer on the concatenation:

$$H_1 = q\left(\left[X_{\alpha_1}, \cdots, X_{\alpha_k}\right] W_{concat} + b_{concat}\right), \tag{13.6}$$

with $W_{concat} \in \mathbb{R}^{(l \cdot k) \times h}$ being a third-order tensor and $b_{concat} \in \mathbb{R}^h$. However, this model might be vulnerable to overfitting due to the high number of parameters.

Independent Weights. To find a better trade-off between model complexity and expressiveness, our third proposed model, *LD_ INDP*, processes each scale independently and combines compact representations in subsequent layers:

$$H_1 = q\left(\left[X_{\alpha_1} W_1 + b_1, \cdots, X_{\alpha_k} W_k + b_k\right]\right), \tag{13.7}$$

with $W_i \in \mathbb{R}^{l \times \tilde{h}}$, $b_i \in \mathbb{R}^{\tilde{h}}$ and $\tilde{h}$ being the number of hidden neurons per scale. Although this model significantly reduces the amount of parameters compared to the *LD_ CONCAT* model (if $\tilde{h} < h$), the amount of parameters is still rather large because each scale i requires its own weight matrix W_i. Furthermore, this model does not exploit that the characteristic label distributions of classes may be similar over all scales.

Shared Weights. In order to account for similarities in label distributions over different scales, our final model, referred to as *LD_SHARED*, uses a single weight matrix W_{sh} which is shared over all scales:

$$H_1 = q\left([X_{\alpha_1}W_{sh} + b_{sh}, \cdots, X_{\alpha_k}W_{sh} + b_{sh}]\right), \tag{13.8}$$

with $W_{sh} \in \mathbb{R}^{l\times\tilde{h}}$ and $b_{sh} \in \mathbb{R}^{\tilde{h}}$. Despite the assumption about similar label patterns in different localities, similarly to LD_INDP this model can learn different scale combinations in subsequent layer.

The Ada-LLD Algorithm

We start with pre-computing APPR and the corresponding label distributions for a small set of k different scales. The label distributions are then combined in H_1 according to Equations 13.5 – 13.8 and fed to the prediction layer H_2. The final MLP model is trained using *Stochastic Gradient Descent (SGD)* to minimize the *cross-entropy loss*

$$\ell(v_i) = \sum_{j=1}^{l} -y_{i,j} \log P_{i,j}, \tag{13.9}$$

where $P_{i,j}$ is the probability of class c_j for node v_i as predicted by our model.

The main steps of our algorithm are summarized in Algorithm 11.

The pre-computation as well as training steps are both highly efficient and scale to large graphs. Computing APPR for all k scales and all n source nodes requires $O({}^{kn}/_{\alpha\epsilon})$ operations [234]. Computing label distributions requires $O(km)$ operations on average where m is the average number of non-zero entries in an APPR-vector. Due to sparsity, it usually holds that $m << n$. Finally, training with SGD is again in $O(n)$. Thus, the complexity of the whole algorithm is linear in the size of the graph.

Extension: Combinations with Additional Node Embeddings

At this point, we wish to emphasize that the combined label distributions learned by our model are in general complementary to node embeddings based on graph topology or additional node attributes. In fact, the node representations learned by Ada-LLD can simply be combined with such embeddings to improve classification accuracy.

As a first step in this direction we propose a simple combination with topological features. We propose to compute topological embeddings, e.g.,

Algorithm 11 *Ada-LLD*

```
Input: Graph $G = (V, E)$, Label matrix $Y_{train}$, Approximation threshold $\epsilon$,
    Teleportation parameters $\{\alpha_1, \dots, \alpha_k\}$
Output: Trained classifier $f$
 1: // Compute label distributions at each scale
 2: declare $X \in \mathbb{R}^{k \times n \times l}$
 3: for $\alpha_j \in \{\alpha_1, \dots, \alpha_k\}$ do
 4:     declare $X_{\alpha_j} \in \mathbb{R}^{n \times l}$
 5:     for $v_i \in V$ do
 6:         $ld_i \leftarrow Compute\_LD(v_i, \alpha_j, \epsilon)$ (see Algorithm 10)
 7:         $X_{\alpha_j}[i, :] = ld_i$
 8:     end for
 9:     $X[j, :, :] = X_{\alpha_j}$
10: end for
11: // Train classifier with SGD
12: $f = H_2(H_1(X))$
13: $f \leftarrow SGD(f, Y_{train}, \ell)$
14: return $f$
```

by simply multiplying an embedding matrix $E \in \mathbb{R}^{n \times d}$ (where d is the dimensionality of the embedding) to the pre-processed adjacency matrix $\hat{A}$ as in [142]. Note that the learned topological embeddings consider only direct neighbors. Those embeddings can be fused into our model by simply concatenating them to the hidden layer H_1. In particular, we choose the most simple of our model variants, *LD_AVG*:

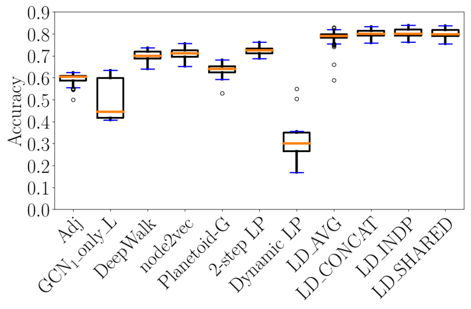
$$H_1 = q\left(\left[\hat{A}E, (\gamma_1 X_{\alpha_1} + \dots + \gamma_k X_{\alpha_k})W_{avg} + b_{avg}\right]\right). \tag{13.10}$$

The embedding matrix E is initialized randomly and learned together with the rest of the model.

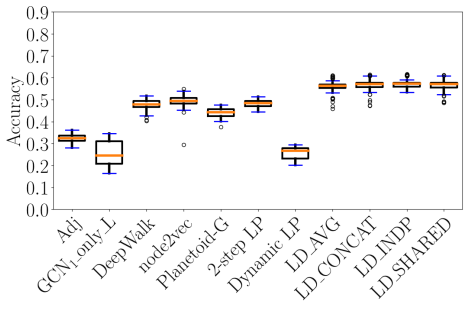
Despite being simple, the above model is already sufficiently expressive to demonstrate how combinations of Ada-LLD with other types of node features can improve classification accuracy. More sophisticated combinations as well as incorporating additional node attributes will be subject to future work.

13.4 Evaluation

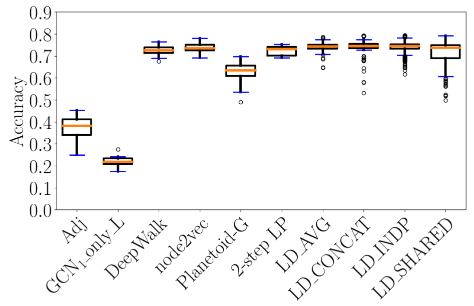
We evaluate our approach by performing node classification and compare the quality in terms of micro F_1 scores, which corresponds to accuracy for multiclass prediction tasks, respectively micro F_1 and macro F_1 scores for



(a) Accuracy scores for *Cora*.



(b) Accuracy scores for *CiteSeer*.



(c) Accuracy scores for *Pubmed*.

Figure 13.2: Accuracy scores for the three benchmark data sets.

multilabel prediction tasks, against state-of-the-art methods. For both tasks, we compare our models against the following approaches:

- *Adj*: a baseline approach which learns node embeddings only based on the information contained in the adjacency matrix
- *GCN*$_1$*_only_L*: a GCN model which applies convolution on the label matrix. We use one convolution layer with the adjacency matrix without self-links, followed by a dense output layer [1]
- *GCN*$_2$: the standard 2-layer GCN as in [142] without using the node attributes
- *DeepWalk*: the DeepWalk model as proposed in [214]
- *node2vec*: the node2vec model as proposed in [103]
- *Planetoid-G*: the Planetoid variant which does not use information from node attributes [271] [2]

[1]We use only a single convolution layer due to the reason stated in Section 13.2

[2]Unless stated differently, we use for all competitors the parameter settings as sug-

For the multiclass problems, we additionally compare against two label propagation approaches, i.e., *2-step LP*, the two-step label propagation approach proposed in [208].

Furthermore we study the benefits of combining label distributions from multiple scales and the effect of combining our embeddings with homophily based embeddings (13.4). We also analyze whether our label based approach is competitive to methods which additionally take node attributes into consideration. Finally, we show that the superiority of our methods is due to the high adaptivity to local label neighborhood.

Multiclass Prediciton

Experimental Setup

We use the following three text classification benchmark graph datasets [229, 191]:

- Cora. The Cora dataset contains 2'708 publications from seven categories in the area of ML. The citation graph consists of 2'708 nodes, 5'278 edges, 1'433 attributes and 7 classes.
- CiteSeer. The CiteSeer dataset contains 3'264 publications from six categories in the area of CS. The citation graph consists of 3'264 nodes, 4'536 edges, 3'703 attributes and 6 classes.
- Pubmed. The Pubmed dataset contains 19'717 publications which are related to diabetes and categorized into 3 classes. The citation graph consists of 19'717 nodes, 44'324 edges, 500 attributes and 3 classes[3].

For each graph, documents are denoted as nodes and undirected links between documents represent citation relationships. If node attributes are applied, bag-of-words representations are used as feature vectors for each document.

We split the data as suggested in [271], i.e., for labeled data our training sets contain 20 randomly selected instances per class, the test sets consist of 1'000 instances and the validation sets contain 500 instances, for each method. The remaining instances are used as unlabeled data. For comparison

gested by the corresponding authors. Except for minor adaptations, e.g., to include label information in the one layer GCN model or to make the Planetoid models applicable for multilabel prediction tasks, we use the original implementations as published by the corresponding authors.

[3]This turned out to be too large for the *Dynamic LP* approach and therefore we cannot show the results of *Dynamic LP* on the Pubmed network.

we use the prediction accuracy scores which we collected over 10 different data splits.

Since the number of iterations for sampling the graph contexts and the label contexts for *Planetoid* are suggested only for the *CiteSeer* data set, we adapted these values relative to the number of nodes for each graph. For *node2vec*, we perform grid searches over the hyperparameters p and q with $p, q \in \{0.25, 0.5, 1.0, 2.0, 4.0\}$ and window size 10 as proposed by the authors. For all models except *Planetoid* unless otherwise noted, we use one hidden layer with 16 neurons, the learning rate and training procedure are used as proposed in [142]. Regarding our models, we use $\alpha \in \{0.01, 0.5, 0.9\}$ as values for the teleportation parameter and $\epsilon = 1e^{-5}$ as approximation threshold to compute the APPR vectors for each node. For *LD_ CONCAT*, *LD_ INDP* and *LD_ SHARED* we use 16 hidden neurons per APPR matrix in the hidden layer.

Results

Figure 13.2 shows boxplots depicting the micro F_1 scores we achieved for the multiclass prediction task for each considered model on the three benchmark datasets *Cora*, *CiteSeer* and *Pubmed*.

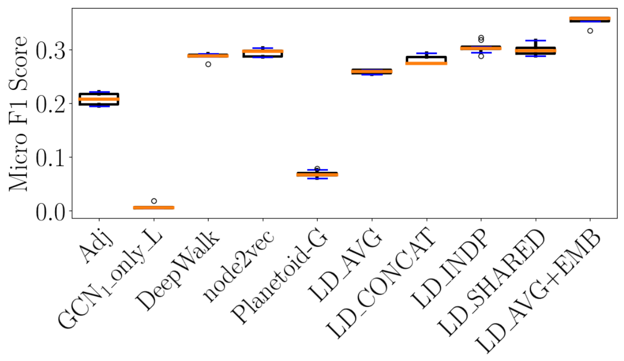
Our models improve the best results produced by *node2vec* which demonstrates that the label distributions are indeed a useful source of information, although the evaluation for GCN_1*_ only_ L* shows, especially for *Pubmed*, rather poor results. This is due to this model considering only the label distribution of a very local neighborhood (in fact one-hop neighbors). However, collecting the label distribution from more spacious neighborhoods gives a significant boost in terms of prediction accuracy. For the *Cora* network, the gain of accuracy of *LD_ INDP*, when comparing to the result of *node2vec*, is more than 13%. Moreover, for all datasets, our models perform similarly, which shows that even simple models with shared weights are able to match the performance of more complex models.

Multilabel Classification

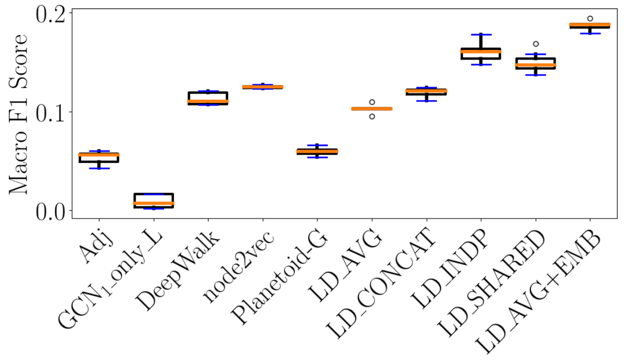
Experimental Setup

We also perform multilabel node classifications on the following two multilabel networks:

- BlogCatalog [245]. This is a social network graph where each of the 10,312 nodes corresponds to a user and the 333,983 edges represent the



(a) Micro-F$_1$ scores for *BlogCatalog*.



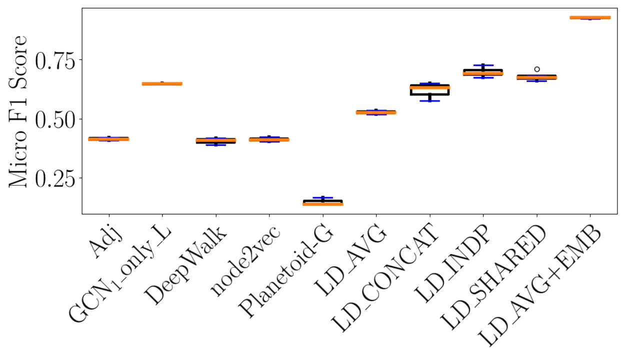
(b) Macro-F$_1$ scores for *BlogCatalog*.

Figure 13.3: Micro F$_1$ and macro F$_1$ for *BlogCatalog*.

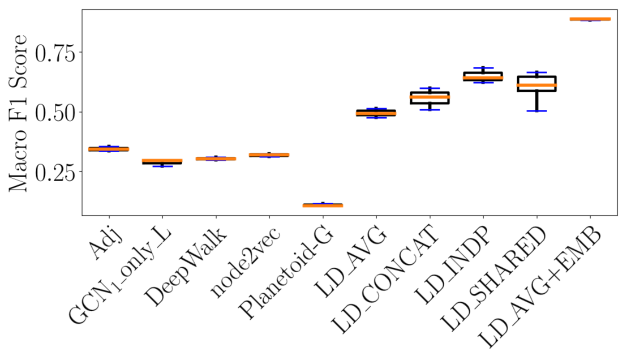
friendship relationships between bloggers. 39 different interest groups provide the labels.

- IMDb Germany. This dataset is taken from [91]. It consists of 32,732 nodes, 1,175,364 edges and 27 labels. Each node represents an actor/actress who played in a German movie. Edges connect actors/actresses that were in a cast together and the node labels represent the genres that the corresponding actor/actress played.

Since the fraction of positive instances is relatively small for most of the classes, we use weighted cross-entropy as loss function. Therefore, the loss caused by erroneously classified positive instances is weighted higher. We use weight 10 in all our experiments. For the same reason we report micro F$_1$ and macro F$_1$ score metrics to measure the quality of the considered methods. We compare our models to the attribute-less models that we already used



(a) Micro-F$_1$ scores for *IMDb Germany*.



(b) Macro-F$_1$ scores for *IMDb Germany*.

Figure 13.4: Micro F$_1$ and macro F$_1$ for *IMDb Germany*.

for the multiclass experiments [4].

We split the data into training, validation and test set such that 70% of all nodes were used for training, 10% for validation and 20% of the data were used to test the model. Note that we could not use stratified sampling splits for these experiments since we optimize for all classes simultaneously instead of using one-vs-rest classifiers [5]. The hyperparameter setting is as described above. For this set of experiments we ran each model, except for *Planetoid-G*, 10 times on five different data splits. Due to the long runtime of *Planetoid-G* we trained this model only three times on two data splits.

[4]To adapt the *Planetoid-G* implementation for multilabel classification, we use a *sigmoid* activation function at the output layer and also slightly changed the embedding learning step. Entities which are used as context and have the same labels as the node itself are sampled from all classes to which the node belongs to.

[5]That is why our results for *node2vec* and *DeepWalk* on the BlogCatalog network are slightly worse than reported in [103]

Results

The results for the *BlogCatalog* graph are shown in Figure 13.3. For this network, only using the label information from the direct neighborhood of a node is not useful to infer its labels, c.f. GCN_1*_ only_ L*. However, incorporating label distributions of somewhat larger neighborhoods as for our models (again, we also use the APPR matrix calculated for small values of α to determine the label distribution in neighborhoods that span more than 1-hop neighbors) seems to improve the results for the prediction task significantly.

Compared to *node2vec* and *DeepWalk* our models achieve similar performance in terms of micro F1 score and clearly outperform them when considering the macro F1 scores. In fact, regarding the median, *LD_ INDP*, which is the best among our models, achieves a gain of about 15.8% over the best competitor, i.e., *node2vec*.

For the *IMDb Germany* network, for which the results can be seen in Figure 13.4, the labels in the 1-hop neighborhood are already very expressive. In this setting, our models outperform all of the competitors. Considering the macro F1 scores, our best model has a median score at about 64% while the best competitor that is *Adj* achieves a score of approx. 34.5%. This results in a gain of more than 85%.

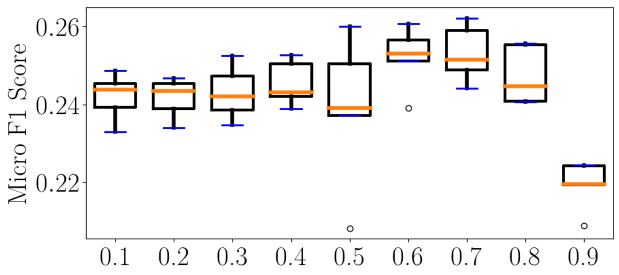
Effect of Combining Multi-Scale Neighborhoods

Figure 13.5 shows how the volume of the considered neighborhood affects the performance of the different models. As an example, we took the *BlogCatalog* network. We ran the label distribution models using only a single APPR matrix with $\alpha \in \{0.1, 0.2, 0.3, \ldots, 0.8, 0.9\}$ to determine the label distribution within the corresponding neighborhood. To show how combining label distributions from multiple neighborhoods affects performance, we furthermore compare the medians of the resulting micro F1 scores to the one achieved by using the combination of only three APPR matrices, i.e., for $\alpha = [0.01, 0.5, 0.9]$.

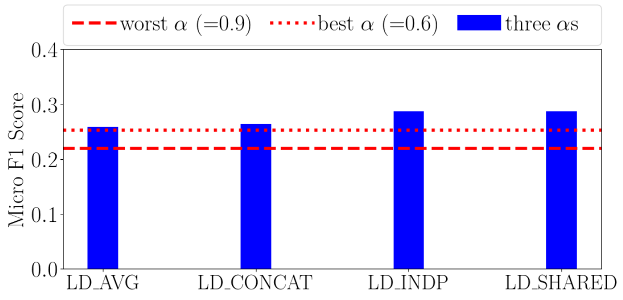
Figure 13.5(a) shows the micro F1 we achieved when running our base model, i.e.,

$$H_1 = q(X_\alpha W),$$

the model that uses only a single neighborhood's label distribution X_α, with a single value of α. Note that H_1 again describes the first hidden layer with q being the activation function (we use $ReLU$). The results show two things: (1) that the native label distribution embeddings are quite sensitive to the choice of α, and (2) that in addition to selecting a single optimal scale, combining multiple scales further improves performance.



(a) Micro F1 scores for single α values.

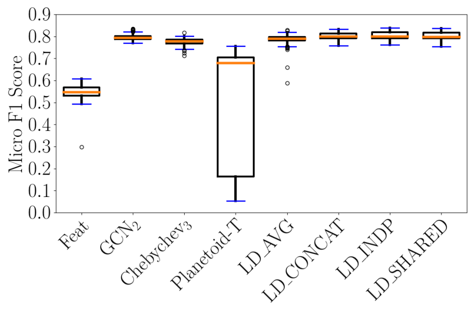


(b) Comparing micro F1 scores of our models when using single α values (dashed lines) vs combining three different α values (bars).

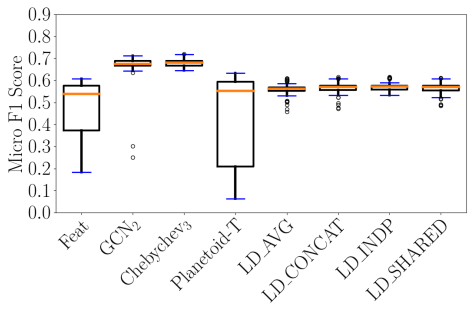
Figure 13.5: Effect of differently sized neighborhoods for label distribution models on the BlogCatalog network.

As can be seen in Figure 13.5(b), due to the models being able to effectively combine differently sized neighborhoods, we can, even with only three different values for α, overcome the micro F1 scores that were achieved when using a single value for α. In addition to the results of the combined models, the dashed lines show the results achieved for single α values. The upper line is the best result (that we got for $\alpha = 0.6$), and the lower line depicts the worst result (which we got for $\alpha = 0.9$).

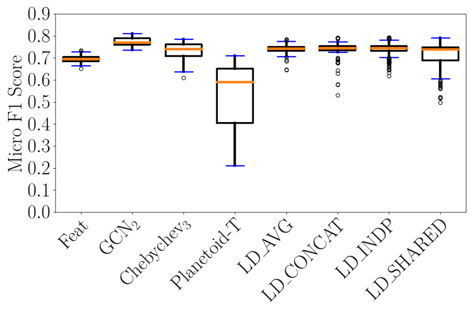
Based on our findings, we can summarize that in cases where no a priori knowledge about "good" and "bad" locality levels is given for a certain dataset, even a small set of α values which range from "very local" (α close to 1) to "spacious" (α close to 0) is already sufficient to get useful node representations due to the models optimizing the combination of neighborhoods.



(a) Accuracy scores for *Cora*.



(b) Accuracy scores for *CiteSeer*.



(c) Accuracy scores for *Pubmed*.

Figure 13.6: Comparison against attribute-based methods: Accuracy scores for the three benchmark data sets.

Combining Ada-LLD Embeddings with Further Node Embeddings

Regarding Figure 13.3 and Figure 13.4 again, we also show the results when combining the node embeddings based on local label distributions with embeddings that capture structural properties. Precisely, the line labeled as *LD_ AVG+EMB* shows the results when using the model described in Equation 13.10.

Even by using this simple extension the performance of our approach is further improved. For the IMDb network, we can observe a noteworthy improvement of accuracy ($> 15\%$) when combining information from label distributions and graph topology, even when using our weakest performing model in this setting. Given these results, we summarize that combining representations based on local label distributions with additional node features is indeed a promising direction for future work.

Comparison to Attribute-Based Methods

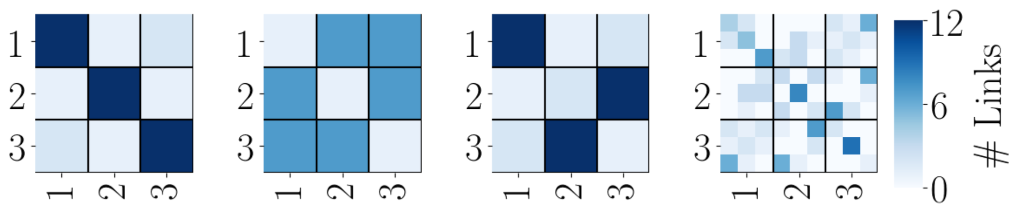
To further investigate the utility of label information compared to that of additional node attributes, we also compare our models against the following state-of-the-art attribute-based methods:

- *Feat*: a baseline approach which uses node attributes as features and predicts node labels only based on these features without considering the underlying graph structure.
- *GCN*$_2$: the standard 2-layer GCN as in [142]
- *Chebychev*$_3$: the spectral convolution method which uses chebychev filters as presented in [80]; as in [142] we also use 3rd order chebychev filters
- *Planetoid-T*: the semi-supervised Planetoid framework which uses attribute information as proposed in [271].

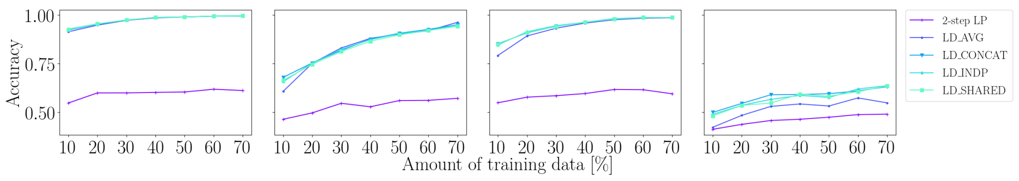
For this set of experiments, we again perform multiclass prediction on the three benchmark text classification datasets and report the prediction accuracy in terms of micro F_1 scores to measure the quality of the retrieved node representations. Note that in contrast to the competitors, our model still does not make use of the node attribute information being available in the given data sets. The results are depicted in Figure 13.6 and clearly show that our models can compete with the attribute-based methods even though they rely only on the provided class labels and do not have access to additional attribute information. Thus, the proposed label-based embeddings yield an expressive alternative in cases where no node attributes are available.

Different Label Patterns

To prove the intuition that our method actually adapts to different local label patterns, we use synthetic networks. Analogously to [208] we apply the stochastic block model (SBM) [118] to generate networks with 8000 nodes and mean degree 15. We define three types of nodes (classes) for each network. The relationships between different classes can be seen in Figure 13.7. The first network models homophily relations, the second models heterophily, the third models both types of relationships for different classes (mixed), and for the fourth graph, each class type is subdivided into different groups to model locally mixing patterns of heterophily and homophily (local variation). Obviously, this is the most difficult setting for all methods. We compare our approaches to *2-step LP* [208], which can model heterophily and homophily.



(a)



(b)

Figure 13.7: Block interaction matrices and corresponding results for different amounts of training instances. The color of the blocks denotes the average number of edges linking nodes from class i to class j.

As can be seen in Figure 13.7, our approaches clearly outperform *2-step LP* on all networks. Thus, even if all classes follow the same pattern it is beneficial to *learn* how labels relate to each other rather than modeling it explicitly. It is particularly interesting that for the most difficult case our simplest model *LD_AVG* performs worse than our other models, which explicitly capture label distributions at multiple scales. Therefore we conclude that considering different proximity is especially useful if labels do not follow a single pattern.

13.5 Conclusion

In this chapter, we presented Ada-LLD, a novel label-based approach to semi-supervised node classification in graphs. Our method aims at learning general correlations between a node's label and neighboring labels in an adaptive fashion. To detect such correlations at multiple scales, we propose four different variants of our model with different inductive biases. The corresponding algorithm is highly efficient and scales to large graphs. Our experimental results on various real-work datasets demonstrate that local label distributions are able to significantly improve node classification in the multi-class as well as in the multi-label setting. We further demonstrated how the success of our method can be explained by its ability to learn local label patterns over different locality levels. As additional experiments, we compared the utility of our label-based features with that of additional node attributes and demonstrated the adaptivity on graphs with different label relationships. In future work, we plan to further investigate how to effectively combine label-based features with different other kinds of features, such as node attributes, edge attributes or node embeddings in a semi-supervised model, to further improve node classification.

Chapter 14

Application of Node Embeddings for Map Fusion

As graph convolutional networks, or more general the message passing framework, have recently been successfully applied on various tasks for graph based data, we finally present an application of node embeddings trained within the message passing framework for the task of Map Fusion. However, we want to mention that the proposed method follows a supervised learning regime. The work presented in this chapter has been accepted and presented at the 2019 NeurIPS Workshop on Graph Representation Learning Workshop.

14.1 Introduction

In recent years the high relevance of graph-structured data has generally been accompanied by an increased demand for algorithms that are able to take advantage from the rich body of relational information encoded in graphs. However, in many domains, the entire knowledge base of a certain domain is spread across multiple data sources, e.g., geo-spatial information are spread across various map providing services, social information are spread across multiple social networks, or, somewhat more general, knowledge bases are distributed across various knowledge graph databases.

In this chapter, we tackle the problem of knowledge fusion by developing a graph neural network model that is able to fuse graph-structured data by learning node matchings. Given two partly overlapping graphs, our approach aims at identifying nodes from both graphs that match to each other by leveraging the local neighborhoods of the nodes. The ultimate goal is to align the graphs such that the information contained in both graphs can be fused properly.

In general, the node matching problem is of high relevance in many applications including the fusion of road networks, also called map fusion [221], the matching of knowledge graphs [240, 210, 277, 106, 250, 283, 261, 266, 64], the alignment of social networks [171], or to enable efficient graph comparison [164, 26]. Traditional methods addressing the graph alignment problem typically rely on calculating distances between manually engineered node and edge features. The distances are used to generate pairs of nodes that serve as matching candidates and the set of candidates is subsequently optimized by incorporating the nodes' neighborhood information [221, 270]. However, those approaches require non-trivial, manual paramater tuning, do not generalize well to unseen data and suffer from high complexities which makes them impractical. More recent works have applied graph neural networks (GNNs) which have proven to be particularly suitable for the graph alignment task [261, 266, 64, 164, 26]. The common approach is to aggregate nodes from the local neighborhood into a target node's representation and subsequently compare the resulting representations with each other. More advanced approaches even additionally aggregate nodes from the other graph into the node representations [164, 266].

Either way, the aggregation of information, i.e., the kind of information as well as the locality of the area from which the information is aggregated, is the critical point to get useful embeddings. This holds for the graph alignment task in particular, as information gathered from local node neighborhoods generally tends to become less useful the more the two graphs that shall be aligned differ in terms of structural properties. To overcome the issue of aggregating irrelevant or even misleading information, state-of-the-art approaches use different types of attention mechanisms when aggregating node neighborhoods. The general idea behind using attention is to determine the importance of a certain entity (e.g., a node in the neighborhood) based on an object's own representation (e.g., the representation of the target node) and the current representation of the entity. The higher the importance, the more influence should the corresponding entity have on the object's own representation. However, the attention based on the structure of the own graph is of limited usefulness for the graph alignment task. We argue that the most important information about a node's neighbors is whether they have good alignments in the counterpart graph.Therefore, the Graph Alignment Network (GrAN) presented in this work uses an importance mechanism that, in contrast to previous works, aims at putting special emphasis on nodes that have a good match in the counterpart graph. By doing this, our model effectively introduces an additional inductive bias, i.e., the assumption that neighboring nodes which are likely to be part of the overlapping area of the two graphs are particularly useful for a target node's representation. The intuition is that matching nodes in the counterpart graph also put special emphasis on nodes whose representations tend to be similar to the representations of the nodes in the source node's neighborhood. Hence, the aggregated information of the two matching nodes in both graphs are likely to be similar, too.

After presenting our GrAN model in Section 14.2, we present preliminary results of our evaluation in Section 14.3. Precisely, we compare our model against state-of-the-art GNN approaches on map fusion tasks. These tasks turned out to be particularly challenging due to presence of geo-spatial coordinates as they form a natural and very strong baseline. However, it is noteworthy that our model can also be applied to other tasks including knowledge graph matching and the determination of graph similarities.

14.2 Graph Alignment Networks with Node Matching Scores

Let $G = (V, E, X, P)$ and $G' = (V', E', X', P')$ denote two graphs with V, V' denoting the sets of nodes, E, E' being the sets of edges and X, X', and P, P' being the node and edge attributes, respectively. For the sake of simplicity we assume that both graphs are undirected. Given this setup, we aim at learning a function $F : G \times G' \rightarrow H \in \mathbb{R}^{|V| \times h}$ which takes two graphs as input, applies multiple message passing operations on them and finally retrieves latent vector representations for the nodes of the first graph that was fed as input into F. For the GrAN model, we use a Siamese architecture that allows to apply the same function F to both graphs such that the model's final output are two node embedding matrices $H = F(G, G')$ and $H' = F(G', G)$, respectively. Given the node embeddings, we subsequently align two nodes if their vector representations are considered similar with respect to some similarity function sim and a predefined similarity threshold τ, i.e.,

$$\hat{y}(v_i, v_j') = \begin{cases} 1 & if \quad sim(\mathbf{h}_i, \mathbf{h}_j) > \tau \\ 0 & else. \end{cases}, \tag{14.1}$$

with $v_i \in V$, $v_j' \in V'$, h_i denoting the row vector from H that corresponds to the embedding of node v_i and h_j being the row vector from H' that corresponds to the embedding vector of node v_j'. $\hat{y}(v_i, v_j') = 1$ indicates that the nodes v_i and v_j' are aligned.

Graph Alignment Network. Graph Neural Networks [224, 98] compute node representations by propagating information between vertices and aggregating them iteratively. In each layer, a message is formed from each source node and passed to its neighbors. Incoming messages for each target node are aggregated and flow into the target node's output representation. Stacking multiple propagation layers allows propagating information over multiple hops. Given a graph $G = (V, E, X, P)$ we define a single message passing propagation step as follows:

$$\begin{aligned} m_{j\to i} &= M^l(\mathbf{h}_j^l, p_{ji}),\ with\ p_{ji} \in P, && (14.2)\\ M^l(\mathbf{h}_j^l, e_{j,i}) &= f_{message}(\mathbf{h}_j^l) + LSTM(p_{ji}), && (14.3)\\ \mathbf{h}_i^{l+1} &= [f_{node}(\mathbf{h}_i^l), \sum_{j,(j,i)\in E} \alpha_{j\to i}^l m_{j\to i}] && (14.4) \end{aligned}$$

where, l stands for the current layer, M is a message function, f_{node} and $f_{message}$ are small neural networks and $[\cdot]$ denotes the concatenation operation. Also, note that for the Map Fusion task, the edge features $p_{ji} \in P$ are sequences of coordinates that represent the road segment. Therefore, we process edge features with a recurrent LSTM network and add the resulting vector to the transformed node representations. However, the inclusion of edge features is generally optional. The weight $\alpha_{j\to i}^l$ determines the importance of the message emitted by node j. In fact, the α weights realize the additional inductive bias of our model, i.e., that the effect that a message has on other nodes is determined by the maximal matching similarity of the source node to the other nodes in other graph. Intuitively, assume that node $v \in V$ has two neighbors $u, w \in V$ with u being aligned to node $u' \in V'$ while node w has no matching node in V'. In this case, our goal is to put high emphasis on u and low emphasis on w when aggregating incoming messages of v. The intuition is that a node $v' \in V'$ who has u' as a neighbor also receives highly weighted information from u' which is assumed to be similar to the information from u. Therefore, in each layer, our approach first determines the best matching for each node with respect to the similarity function sim, and all outgoing messages are weighted accordingly. More formally, we define the message weight $\alpha_{j\to i}^l$ as follows:

$$\begin{aligned} \alpha_{j\to i}^l &= \frac{exp(I(\mathbf{h}_j^l, H'^l))}{\sum_{\hat{j}} exp(I(\mathbf{h}_{\hat{j}}^l, H'^l))},\ with\ (\hat{j}, i) \in E,\ and && (14.5)\\ I(\mathbf{h}_j^l, H'^l) &= \max_j sim'(f_{match}(\mathbf{h}_j^l), f_{match}(\mathbf{h'}_k{}^l)),\ with\ \mathbf{h'}_k{}^l \in H'^l. && (14.6) \end{aligned}$$

Since it is not possible to backpropagate through the max operation, we apply the gumbel softmax trick [178, 129]. For the functions $f_{message}$, f_{node} and f_{match}, we use Multilayer Perceptrons and add an additional inductive bias by sharing parameters between these three functions. The inverse Euclidean function is used as similarity function $sim'(a, b) = \frac{1}{d_{euclidian}(a,b)+1}$.

Learning. We use the contrastive loss [107] with positive ϵ^+ and negative ϵ^- margins, i.e.,

$$L = y_{i,j} \cdot max(0, d(\mathbf{h}_i, \mathbf{h}_j) - \epsilon^+) + (1 - y_{i,j}) \cdot max(0, \epsilon^- - d(\mathbf{h}_i, \mathbf{h}_j)), \quad (14.7)$$

to train our model. $y_{i,j}$ is 1 for aligned and 0 for non-aligned pairs of nodes, and d is the Euclidean distance. In contrast to, e.g., triplet loss, the contrastive loss allows us to incorporate nodes having no matchings into the training.

14.3 Experiments

We evaluate our approach by interpreting the node matching problem as a binary classification task and compare the proposed GrAN model against several state-of-the-art methods including GCN [142], GAT [256], and GraphSAGE [108]. All GNN models use the same siamese architecture with shared weights. For the competitors, as well as for our model we use implementations from the pytorch geometric framework [95]. We also evaluate against a baseline method where we simply use the geo-spatial coordinates of the used road networks' vertices as 2-dimensional node representations for the evaluation. Note that this is a fairly strong baseline as the graph vertices lie within a continuous space and two matching nodes obviously tend to have similar geo-spatial coordinates. Additionally, where it is possible, we report the results when using probabilistic relaxation [270]. Note that the probabilistic relaxation method considers all assignments at once using hungarian algorithm [189] in combination with Otsu's [199] method, and therefore can benefit from the fact that graphs are different in size. However, the complexity of $O(c^3)$, with c being the number of matching candidates, is rather high and hence makes it impractical for larger networks. In contrast, all GNN models classify each candidate pair independently. To achieve a fair comparison, we additionally report results when conducting nearest neighbor queries on the set of nodes from the larger graph and query objects only being taken from the smaller graph. This way we still need to find a threshold to divide the retrieved candidates into nodes with and without matchings, but consider the fact that graphs differ in size. Note that this approach ignores cases where nodes match to multiple nodes in counterpart graph.

We train each of the models by using two different road networks (one is from OSM[1] and the other from a commercial map provider), that both cover freeways of the Northrhine-Westphalia region in Germany. In total,

[1]https://www.openstreetmap.org

	NRW (train)	NRW (test)	MUC (test)
Nodes in G_1	2,430	308	2,996
Nodes in G_2	9,924	1,206	3,153
Candidates	52,138	6,148	378,948
1:1 & 1:n matchings	2,359	294	1,777
1:0 matchings	99	3	1,257

Table 14.1: Training and test dataset statistics.

the networks consist of 15,442 vertices and 20,166 edges with 2956 of the vertices having a matching node in the other network, i.e., they form matching pairs. In general, nodes can have zero (1:0), one (1:1) or multiple (1:n) matching nodes in the other graph. Every node represents an intersection and each edge corresponds to a road segment represented by the sequence of coordinates. To train the models, the matching pairs are split into training, validation and test set (80-10-10). For the purpose of generating negative samples, we project each of the positive sample vertices into the other road network and perform spatial range queries with a radius of approximately 1km[2]. The resulting set of vertices form the negative samples of the corresponding node (except for the actual matching vertex). Additionally, we perform the same range queries for the nodes labeled as 1:0 matchings and add the resulting pairs to the negative samples. Finally we get a set of 65,432 positive and negative candidate pairs, that form the final training, validation and test sets as reported in Table 14.1. Furthermore, we evaluate how well the models generalize to yet unseen road networks by measuring their matching performance on a pair of road network sections that are taken from the urban area of the city of Munich, Germany. Beside freeways, these graphs also contain types of roads that are not present in the training data, e.g., residential street. The dataset is referred to as MUC.

For all models we trained various different architectures whose details and hyperparameter settings can be found in the appendix of this chapter. In the following, we report the results for the architectures that showed the best performance. The results reported for the GrAN model were achieved when using the Adam SGD optimizer and four message propagation layers. After each convolutional layer, we update the output node representation by combining it with the messages received from the neighborhood and the encoding of the corresponding edge geometries. For the subsequent classification and as similarity function for the node encoding we use the Euclidean distance.

[2]Note that some matching nodes representing the same acceleration lanes or freeway exits are indeed located that far from each other.

The contrastive loss has been used as loss function to be able to incorporate negative samples during training.

Method	**NRW**	**MUC**
GrAN	.676	.642
GCN	.091	.009
GAT	.091	.009
GraphSAGE	.091	.009
Spatial Coordinates	.681	.626
Probabilistic Relaxation	.752	-

Table 14.2: Resulting F1 Scores for the NRW and MUC datasets when using MLP classifier.

Method	**NRW**	**MUC**
GrAN	.847	.704
GCN	.048	.001
GAT	.071	.005
GraphSAGE	.064	.001
Spatial Coordinates	.827	.661
Probabilistic Relaxation	.752	-

Table 14.3: Resulting F1 Scores for the NRW and MUC datasets when using NN queries.

Table 14.2 shows the F1 scores comparing the classification results of our approach against the results produced by the other GNN based models and when using only spatial coordinates, or the probabilistic relaxation method. While our approach outperforms GCN, GAT and GraphSage, and also the spatial coordinates baseline on the MUC dataset, we achieve similar matching results as the baseline on the NRW data. When comparing against probabilistic relaxation, the latter achieves better results on the NRW dataset. However, considering the nearest neighbors evaluation (cf. Table 14.3), the probabilistic relaxation approach is outperformed by both our approach and the spatial coordinates baseline, although all methods except for probabilistic relaxation ignore 1:n matchings. In summary, our method still achieves best results on both datasets.

14.4 Conclusion

In this chapter we studied the problem of graph alignment. We presented our ongoing work on a new Graph Neural Network based model, that aggregates information from neighbors based on their alignment scores. We evaluated the proposed approach on Map Fusion tasks and compared it with state-of-the-art models. Our promising experimental results show that the proposed approach outperforms other GNN models by large margins. In future work we plan to adapt our model for other tasks like the alignment of entities in heterogeneous graphs, and also for calculating graph similarities. We further see the potential of improving the runtime of our algorithm by combining it with graph coarsening methods for instance.

Appendix

For all Graph Neural Network models we tried different architectures and hyperparameters:

Weight Sharing in Siamese Architecture We also tried to learn different encoders for each map without weight sharing. This could help if two maps have different biases. However, it just worsened the results in our setups.

Unsupervised Pretraining We used an adapted version of Deep Graph Infomax [257] for pretraining. To corrupt a graph we shifted nodes randomly in the coordinate space in each iteration. In summary, the model pretraining led to the fastest convergence but did not affect the final results significantly.

Loss We experimented with different positive and negative margins. We obtained best results with a positive margin of 0 and a negative margin of 10. To account for the class imbalance, we decreased the weight for the negative examples inversely proportional to the relative frequency.

Model Depth We tried 1 to 8 message passing layers, followed by 0 to 2 fully connected layers. For GAT we additionally tried 1 to 7 attention heads in each layer. The number of attention heads was constant for all layers.

Width We tested various layer sizes, with both constant and variable sizes of layers in the same model. Precisely, the layers consisted of 64, 128, or 256 units each.

Edge Attributes For a fair comparison we adapted the message function of all GNN models to be able to consider edge attributes and we evaluated them with and without using edge attributes. We also trained models which consider edge attributes only in the first layer. By using the Douglas–Peucker algorithm we reduced edge attribute sequences of different lengths to be of equal lengths. We tried the sum and concatenation operations for combining edge and node embeddings in the function M.

Early Stopping We trained all models with early stopping and patience value of 1000.

Message Aggregation For aggregating the messages, we tried mean/max pooling and concatenation.

Normalization We normalized input coordinates to [-1,1] range. Therefore, the data has been zero-centered for each dimension at first. Next, each dimension was divided by the max distance from the center in all dimensions. This normalization stabilizes training and inference while preserving distance ratios.

Optimizer Settings All tested models were trained with the Adam SGD optimizer with a learning rate $lr \in [1\text{e-}4, 1\text{e-}2)$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a weight decay value that has been chosen from the interval $[0, 1\text{e-}4)$.

Chapter 15

Concluding Remarks

This thesis mainly concentrates on unsupervised learning techniques suitable for analyzing and processing data as they could appear in the field of social data analytics. Since the analysis of social data includes many different facets of challenges, e.g., their temporal dynamics, high dimensionality or complex nature due to relational information, it is the research in this area that is particularly demanded. Generally, the evolving tasks and steadily growing amount of data always require for novel techniques that are able to cope with the upcoming problems. This became especially concrete in the field of social data science, as social media platforms became part of daily life and economy as well as politics recognized the huge potential captured within the revealed information. However, the same trend can be observed in other sectors than social data science, too. For instance, very similar challenges appear when considering the automotive or manufacturing sectors where vehicles or machines are equipped with more and more sensors to support automation. To tackle some of the upcoming challenges, this thesis presents some modern, respectively renewed data science algorithms that are suitable to deal with these modern problems.

In the first part of the thesis, we started by presenting two algorithm that address the problem of finding clusters in high-dimensional data that may change over time. Precisely, those algorithms detect arbitrarily oriented subspace clusters in data streams. While previous works in this research field mainly focused on identifying clusters in axis-parallel subspaces, the proposed methods have the advantage of being able to combine certain features such that correlations within the data are exploited to compress the captured information. The first algorithm is based on PCA and is biased towards finding local cluster patterns while the second one follows a global approach that is based on Hough transformation. Also, both algorithms follow different goals that are achieved by employing different data streaming models. The

PCA-based method primarily aims at reducing the runtime, and hence we proposed to use a continuous online phase that compresses the current state of clusters within generic microcluster structures. The final clustering model can subsequently be determined on demand by triggering an offline phase for which one can basically use any static PCA based correlation clustering algorithm. The streaming algorithm that is based on Hough transformation uses a different streaming model, i.e., a batched processing model, that aims at reducing the required memory consumption. As the static counterpart algorithm has a rather high complexity that highly depends on the size of the database, we achieved to reduce the required memory remarkably by adapting the processing scheme accompanied by proper aggregation and merging strategies. However, due to the fact that these (as well as the static) correlation clustering methods cluster based on latent features that are formed during the dimensionality reduction, these methods generally lack in interpretability. From a practical point of view, we therefore see the need of making those algorithms, respectively the results of those algorithms, more explainable in future work. Moreover, as neural subspace clustering methods that identify subspace cluster based on the self-expressiveness assumption recently showed promising results for axis-parallel subspace clustering, another direction for future work would be the development of suitable algorithms that learn to detect arbitrarily oriented subspace clusters. Furthermore, we also presented our study on identifying users of social media platforms based on their mobility patterns. Beside revealing the potential of using such information for fusing even anonymized social databases, we also identified privacy issues that in turn yield the challenge of developing proper methods for more powerful anonymization techniques. Finally, the first part also includes a work that envisions the computation of social maps at different scales by using microblog data in combination with hierarchical clustering approaches. Ultimately, these social maps are able to reveal areas that follow similar trends or have the same topics in mind. As the NLP community presented a huge body of successful works on text embedding methods in the meanwhile, it would be interesting to calculate social maps by using sophisticated text representation methods that also take contextual information into account, or work on multi-lingual text corpora.

The second part of the thesis focused on learning algorithms for graph-structured data. In the beginning of this part, we presented an unsupervised method for learning homophily based node embeddings. By using Personalized PageRank vectors as basis to generate context samples for the word2vec-like learning scheme, the proposed method enables to adapt a node's considered neighborhood to the local, structural properties of the graph for each node individually. Consequently, the samples for low degree nodes, that are generally expected to be located in a graph's periphery, consider a more spacious neighborhood for sampling context nodes. In contrast, high degree nodes, typically being located in the center of expander-like graphs for instance, are biased towards considering only close-by neighbors for their context. Secondly, we developed a method with a different inductive bias, such that it generates structure-based node embeddings. By using the entropy values of the stationary personalized PageRank distributions in differently scaled, local node neighborhoods, we already achieved meaningful node descriptors. We subsequently used those node descriptors as input to various clustering/aggregation algorithms. This way we learned abstract node role notions in an unsupervised fashion, and used these role notions to generate vector representations for entire graph structures. After presenting those unsupervised methods, we also addressed a semi-supervised setting for learning models that aim at solving node classification tasks. In this setting we also considered a small portion of labeled nodes in addition to the unsupervised information that is naturally given by the graph topology. Somewhat more precise, we present models that are not limited to the homophily or heterophily assumption, but are able to learn general correlations between a node's label and the labels that are present in that node's local neighborhood. Lastly, we present a work that tackles the general problem of knowledge fusion by learning matching nodes from two partly overlapping graphs in a supervised manner. The identification of matching nodes finally enables to match the two graphs on each other and to fuse the information that is given both graphs. Specifically, we focus on the task of map fusion, but regarding future directions this model can theoretically also be applied for knowledge graph fusion tasks and even more general for determining graph similarities. Further, as the homophily- and structure-based node embedding techniques rely on Personalized PageRank, we also see the potential to learn node embeddings in dynamic settings. In general the context exploration can already be calculated efficiently, but one major challenge is to keep the embeddings up-to-date. As skip-gram models are hardly applicable here, one might consider other models, e.g., the glove model [213] that relies on co-occurrence probabilities.

Lastly, as some of the author's works that have been published during this thesis' work did not make it into the thesis, either because the methods are not directly related to applications on social data or because they do not fall into the direction of learning methods, we want to refer the interested reader to the author's other works. The works presented in [50, 51] address the problem of RkNN query processing, the work presented in [36] presents an approach on neural indexing for the MRkNNCoP-Tree and finally, the work that can be found in [52] is the invited publication for one of the GIS

Cup 2019 winner approaches.

Acknowledgements

This thesis would not have been possible without the support and encouragement of the people around me. I am very grateful for the professional and personal support I received from them during the past few years. My most profound thanks are dedicated to all of them.

I would particularly like to thank my supervisor and first referee for this thesis, Prof. Dr. Peer Kröger. Not only did he fund my position initially, but he also taught me a lot of skills that have been required for this work, inspired me with his optimism, and provided me with professional support and sound technical guidance.

Special thanks go to Prof. Dr. Emmanuel Müller for his interest in my work and his willingness to act as the second referee for this thesis.

I am also especially thankful to Prof. Dr. Thomas Seidl who also provided funding for my position and, as head of the chair for database systems and data mining, is responsible for creating the basic conditions for the good and constructive atmosphere within the group.

My deep gratitude obviously goes to all of my colleagues, too. They have made the past few years a great experience and a pleasant part of my life. Special thanks go to Prof. Dr. Matthias Schubert, Prof. Dr. Andreas Züfle, Dr. Gregor Jossé, Dr. Johannes Niedermayer, Dr. Tobias Emrich, Julian Busch, Max Berrendorf, Sebastian Schmoll, Daniyal Kazempour, Anna Beer and Sabrina Friedl. In particular, I want to thank Evgeniy Faerman with whom I collaborated a lot. It's been a great pleasure working with you!

Furthermore, I especially thank Susanne Grienberger who always helped with administrative things and Franz Krojer who always provided help with technical issues.

I also want to thank Prof. Dr. Mario A. Nascimento for his supervision during my stay at the University of Alberta, and Prof. Dr. Michael W. Mahoney for supervising me during my stay at UC Berkeley.

Finally, I want to express my very great appreciation to my parents, my siblings and my friends. Your support and encouragement have always been and will forever be invaluable to me.

List of Figures

List of Tables

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, 2016.

[2] Osman Abul, Francesco Bonchi, and Mirco Nanni. Never Walk Alone: Uncertainty for Anonymity in Moving Objects Databases. In *2008 IEEE 24th International Conference on Data Engineering*, pages 376–385. Ieee, IEEE, apr 2008.

[3] Elke Achtert, Christian Böhm, Jörn David, Peer Kröger, and Arthur Zimek. Global correlation clustering based on the hough transform. *Statistical Analysis and Data Mining*, 1(3):111–127, 2008.

[4] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Ina Müller-Gorman, and Arthur Zimek. Detection and visualization of subspace cluster hierarchies. In *International Conference on Database Systems for Advanced Applications*, pages 152–163. Springer, 2007.

[5] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. On exploring complex relationships of correlation clusters. In *Scientific and Statistical Database Management, 2007. SSBDM'07. 19th International Conference on*, pages 7–7, 2007.

[6] Elke Achtert, Christian Böhm, Hans-Peter Kriegel, Peer Kröger, Arthur Zimek, et al. Robust, complete, and efficient correlation clustering. In *SDM*, pages 413–418, 2007.

[7] Elke Achtert, Christian Böhm, Peer Kröger, and Arthur Zimek. Mining hierarchies of correlation clusters. In *Scientific and Statistical Database Management, 2006. 18th International Conference on*, pages 119–128, 2006.

[8] Märtens M. Raupach C. Swierkot K. Lammersen C. Ackermann, M. R. and C. Sohler. Streamkm++: A clustering algorithm for data streams. *Journal of Experimental Algorithmics (JEA)*, 17:2–4, 2012.

[9] A. B. Adcock, B. D. Sullivan, and M. W. Mahoney. Tree decompositions and social graphs. *Internet Mathematics*, 12(5):315–361, 2016.

[10] Aaron B Adcock, Blair D Sullivan, and Michael W Mahoney. Tree-like structure in large social and information networks. In *Proceedings of 2013 IEEE 13th International Conference on Data Mining*, pages 1–10. IEEE, 2013.

[11] Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *Proceedings of the 2018 Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 170–182. Springer, 2018.

[12] C C Aggarwal and P S Yu. Condensation approach to privacy preserving data mining. *Adv. In Database Technology - Edbt 2004, Proc.*, 2992:183–199, 2004.

[13] Charu C Aggarwal. An introduction to social network data analytics. In *Social network data analytics*, pages 1–15. Springer, 2011.

[14] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th international conference on Very large data bases-Volume 29*, pages 81–92, 2003.

[15] Charu C Aggarwal, Jiawei Han, Jianyong Wang, and Philip S Yu. A framework for projected clustering of high dimensional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 852–863. VLDB Endowment, 2004.

[16] Charu C Aggarwal, Joel L Wolf, Philip S Yu, Cecilia Procopiuc, and Jong Soo Park. Fast algorithms for projected clustering. In *ACM SIGMoD Record*, volume 28, pages 61–72, 1999.

[17] Charu C Aggarwal and Philip S Yu. *Finding generalized projected clusters in high dimensional spaces*, volume 29. 2000.

[18] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[19] Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining Text Data*, pages 77–128. 2012.

[20] Rakesh Agrawal, Johannes Gehrke, Dimitrios Gunopulos, and Prabhakar Raghavan. *Automatic subspace clustering of high dimensional data for data mining applications.* ACM, 1998.

[21] Kristen M Altenburger and Johan Ugander. Monophily in social networks introduces similarity among friends-of-friends. *Nature Human Behaviour*, 2(4):284, 2018.

[22] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *Proceedings of 2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

[23] Ira Assent, Ralph Krieger, Emmanuel Müller, and Thomas Seidl. Dusc: Dimensionality unbiased subspace clustering. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 409–414. IEEE, 2007.

[24] Ira Assent, Ralph Krieger, Emmanuel Müller, and Thomas Seidl. Visa: visual subspace clustering analysis. *ACM SIGKDD Explorations Newsletter*, 9(2):5–12, 2007.

[25] James Atwood and Don Towsley. Search-convolutional neural networks. *CoRR*, abs/1511.02136, 2015.

[26] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pages 384–392. ACM, 2019.

[27] Mukund Balasubramanian and Eric L Schwartz. The isomap algorithm and topological stability. *Science*, 295(5552):7–7, 2002.

[28] Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

[29] Alexander Bayarma, Ryuichi Kitamura, and Yusak Susilo. Recurrence of Daily Travel Patterns: Stochastic Process Approach to Multiday Travel Behavior. *Transportation Research Record: Journal of the Transportation Research Board*, 2021:55–63, dec 2007.

[30] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Proceedings of Advances in neural information processing systems*, volume 14, pages 585–591, 2001.

[31] Jose Bento and Stratis Ioannidis. A family of tractable graph distances. In *Proc. of SIAM SDM*, pages 333–341, 2018.

[32] A.R. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Pervasive Computing*, 2(1):46–55, jan 2003.

[33] Pavel Berkhin. Bookmark-coloring algorithm for personalized pagerank computing. *Internet Mathematics*, 3(1):41–62, 2006.

[34] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint 1209.2684*, 2012.

[35] Donald J Berndt and James Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. *AAAI-94 Workshop on Knowledge Discovery in Databases (KDD-94)*, 398:359–370, 1994.

[36] Max Berrendorf, Felix Borutta, and Peer Kröger. k-distance approximation for memory-efficient rknn retrieval. *To appear in Proceedings of the 12th International Conference on Similarity Search and Applications*, 2019.

[37] Claudio Bettini, X. Sean Wang, and Sushil Jajodia. Protecting Privacy Against Location-Based Personal Identification. In Willem Jonker and Milan Petković, editors, *Secure Data Management: Second VLDB Workshop, SDM 2005, Trondheim, Norway, September 2-3, 2005. Proceedings*, volume 3674 LNCS, pages 185–199. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[38] Smriti Bhagat, Graham Cormode, and S Muthukrishnan. Node classification in social networks. In *Social network data analytics*, pages 115–148. Springer, 2011.

[39] C. M. Bishop. *Pattern Recognition and Machine Learning.* 2006.

[40] Andrew J Blumberg and Peter Eckersley. On Locational Privacy, and How to Avoid Losing it Forever. *Electronic Frontier Foundation Tech Rep August*, (August):1–7, 2009.

[41] C. Böhm, K. Kailing, H.-P. Kriegel, and P. Kröger. Density connected clustering with local subspace preferences. 2004.

[42] Christian Böhm, Karin Kailing, Peer Kröger, and Arthur Zimek. Computing clusters of correlation connected objects. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 455–466, 2004.

[43] Aleksandar Bojchevski and Stephan Günnemann. Deep gaussian embedding of attributed graphs: Unsupervised inductive learning via ranking. *arXiv preprint arXiv:1707.03815*, 2017.

[44] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM'05)*, pages 8–pp, 2005.

[45] Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 21(suppl_1):i47–i56, 2005.

[46] Felix Borutta, Julian Busch, Evgeniy Faerman, Adina Klink, and Matthias Schubert. Structural graph representations based on multiscale local network topologies. *To appear in 2019 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, 2019.

[47] Felix Borutta, Julian Busch, Evgeniy Faerman, and Matthias Schubert. Towards learning structural node embeddings using personalized pagerank. In *2017 Proceedings of 2017 Conference on Lernen, Wissen, Daten, Analyen (LWDA)*, page 32, 2017.

[48] Felix Borutta, Thomas Hubauer, and Peer Kröger. Method and system for monitoring sensor data of rotating equipment, 2017. US Patent US10339784 (B2), US Patent App. US2017365155 (A1), EU Patent App. EP3258333 (A1).

[49] Felix Borutta, Thomas Hubauer, and Peer Kröger. A generic summary structure for correlation clustering on data streams. *To appear in Proceedings of the 12th International Conference on Similarity Search and Applications*, 2019.

[50] Felix Borutta, Mario A Nascimento, Johannes Niedermayer, and Peer Kröger. Monochromatic rknn queries in time-dependent road networks. In *Proceedings of the Third ACM SIGSPATIAL International Workshop on Mobile Geographic Information Systems*, pages 26–33. ACM, 2014.

[51] Felix Borutta, Mario A Nascimento, Johannes Niedermayer, and Peer Kröger. Reverse k-nearest neighbour schedules in time-dependent road networks. In *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, page 27. ACM, 2015.

[52] Felix Borutta, Sebastian Schmoll, and Sabrina Friedl. Optimizing the spatio-temporal resource search problem with reinforcement learning. *To appear in Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2019.

[53] Paul S Bradley, Usama M Fayyad, Cory Reina, et al. Scaling clustering algorithms to large databases. In *KDD*, volume 98, pages 9–15, 1998.

[54] Leo Breiman et al. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical science*, 16(3):199–231, 2001.

[55] Bobby-Joe Breitkreutz, Chris Stark, Teresa Reguly, Lorrie Boucher, Ashton Breitkreutz, Michael Livstone, Rose Oughtred, Daniel H Lackner, Jürg Bähler, Valerie Wood, et al. The biogrid interaction database: 2008 update. *Nucleic acids research*, 36(suppl 1):D637–D640, 2008.

[56] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *CoRR*, abs/1312.6203, 2013.

[57] Ceren Budak, Theodore Georgiou, Divyakant Agrawal, and Amr El Abbadi. Geoscope: Online detection of geo-correlated information trends in social networks. *Proceedings of the VLDB Endowment*, 7(4):229–240, 2013.

[58] Ji-Won Byun, Ashish Kamra, Elisa Bertino, and Ninghui Li. Efficient k-Anonymization Using Clustering Techniques. In *Advances in Databases: Concepts, Systems and Applications*, number 0430274, pages 188–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[59] Hongyun Cai, Vincent W Zheng, and Kevin Chang. A comprehensive survey of graph embedding: problems, techniques and applications. *IEEE Transactions on Knowledge and Data Engineering*, 2018.

[60] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *SDM*, volume 6, pages 328–339, 2006.

[61] Feng Cao, Martin Estert, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of the 2006 SIAM international conference on data mining*, pages 328–339. SIAM, 2006.

[62] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM conference on Information and knowledge management*, pages 891–900. ACM, 2015.

[63] Wei Cao, Zhengwei Wu, Dong Wang, Jian Li, and Haishan Wu. Automatic user identification method across heterogeneous mobility data sources. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 978–989. IEEE, may 2016.

[64] Yixin Cao, Zhiyuan Liu, Chengjiang Li, Juanzi Li, and Tat-Seng Chua. Multi-channel graph neural network for entity alignment. *arXiv preprint arXiv:1908.09898*, 2019.

[65] Soumen Chakrabarti, Martin Ester, Usama Fayyad, Johannes Gehrke, Jiawei Han, Shinichi Morishita, Gregory Piatetsky-Shapiro, and Wei Wang. Data mining curriculum: A proposal (version 1.0). *Intensive Working Group of ACM SIGKDD Curriculum Committee*, 140, 2006.

[66] Yixin Chen and Li Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM, 2007.

[67] Chun-Hung Cheng, Ada Wai-Chee Fu, and Yi Zhang. Entropy-based subspace clustering for mining numerical data. In *Proceedings of the 1999 ACM SIGKDD International Conference on Knowledge Discovery in Databases*, pages 84–93. ACM, 1999.

[68] Hao Cheng, Kien A Hua, and Khanh Vu. Constrained locally weighted clustering. *Proceedings of the VLDB Endowment*, 1(1):90–101, 2008.

[69] Zhiyuan Cheng, James Caverlee, and Kyumin Lee. You are where you tweet: a content-based approach to geo-locating twitter users. In *CIKM*, pages 759–768. ACM, 2010.

[70] Yi-Hong Chu, Jen-Wei Huang, Kun-Ta Chuang, De-Nian Yang, and Ming-Syan Chen. Density conscious subspace clustering for high-dimensional data. *IEEE Transactions on knowledge and data engineering*, 22(1):16–30, 2010.

[71] Fan Chung. The heat kernel as the pagerank of a graph. *Proceeddings of the National Academy of Sciences*, 104(50):19735–19740, 2007.

[72] Lawrence Corwin. *Multivariable calculus.* Routledge, 2017.

[73] João Paulo Costeira and Takeo Kanade. A multibody factorization method for independently moving objects. *International Journal of Computer Vision*, 29(3):159–179, 1998.

[74] Thomas M Cover and Joy A Thomas. *Elements of information theory.* John Wiley & Sons, 2012.

[75] David M Cutler and Edward L Glaeser. Are ghettos good or bad? Technical report, National Bureau of Economic Research, 1995.

[76] Manoranjan Dash and Poon Wei Koot. Feature selection for clustering. In *Encyclopedia of database systems*, pages 1119–1125. 2009.

[77] Yves-Alexandre de Montjoye, César A. Hidalgo, Michel Verleysen, and Vincent D. Blondel. Unique in the Crowd: The privacy bounds of human mobility. *Scientific Reports*, 3:1376, mar 2013.

[78] Karsten M Decker and Sergio Focardi. Technology overview: A report on data mining. 1995.

[79] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JAsIs*, 41(6):391–407, 1990.

[80] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Proceedings of Advances in neural information processing systems*, pages 3844–3852. 2016.

[81] Carlotta Domeniconi, Dimitris Papadopoulos, Dimitrios Gunopulos, and Sheng Ma. Subspace clustering of high dimensional data. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 517–521. SIAM, 2004.

[82] Claire Donnat, Marinka Zitnik, David Hallac, and Jure Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1320–1329. ACM, 2018.

[83] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. volume 15, pages 11–15. ACM, 1972.

[84] Richard O Duda, Peter E Hart, and David G Stork. *Pattern classification.* John Wiley & Sons, 2012.

[85] Ehsan Elhamifar and Rene Vidal. Sparse subspace clustering: Algorithm, theory, and applications. *IEEE transactions on pattern analysis and machine intelligence*, 35(11):2765–2781, 2013.

[86] Ahmed K. Elmagarmid, Panagiotis G. Ipeirotis, and Vassilios S. Verykios. Duplicate Record Detection: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1):1–16, jan 2007.

[87] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231, 1996.

[88] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[89] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert. Semi-supervised learning on graphs based on local label distributions. *arXiv preprint arXiv:1802.05563*, 2018.

[90] Evgeniy Faerman, Felix Borutta, Julian Busch, and Matthias Schubert. Semi-supervised learning on graphs based on local label distributions. In *Proceedings of the KDD 14th International Workshop on Mining and Learning with Graphs (MLG)*, 2018.

[91] Evgeniy Faerman, Felix Borutta, Kimon Fountoulakis, and Michael W Mahoney. Lasagne: Locality and structure aware graph node embedding. *arXiv preprint arXiv:1710.06520*, 2017.

[92] Evgeniy Faerman, Felix Borutta, Kimon Fountoulakis, and Michael W Mahoney. Lasagne: Locality and structure aware graph node embedding. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 246–253. IEEE, 2018.

[93] Fredrik Farnstrom, James Lewis, and Charles Elkan. Scalability for clustering algorithms revisited. *SIGKDD explorations*, 2(1):51–57, 2000.

[94] Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI magazine*, 17(3):37, 1996.

[95] Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[96] Jerome H Friedman and Jacqueline J Meulman. Clustering objects on subsets of attributes (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 66(4):815–849, 2004.

[97] Wolfgang Gatterbauer, Stephan Günnemann, Danai Koutra, and Christos Faloutsos. Linearized and single-pass belief propagation. *VLDB Endowment*, 8(5):581–592, 2015.

[98] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1263–1272. JMLR. org, 2017.

[99] Sanjay Goil, Harsha Nagesh, and Alok Choudhary. Mafia: Efficient and scalable subspace clustering for very large data sets. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 443, page 452. ACM, 1999.

[100] Marta C. González, César A. Hidalgo, and Albert-László Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, jun 2008.

[101] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[102] John C Gower. Some distance properties of latent root and vector methods used in multivariate analysis. *Biometrika*, 53(3-4):325–338, 1966.

[103] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[104] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan o'Callaghan. Clustering data streams. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 359–366. IEEE, 2000.

[105] Stephan Günnemann, Hardy Kremer, and Thomas Seidl. Subspace clustering for uncertain data. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 385–396. SIAM, 2010.

[106] Lingbing Guo, Zequn Sun, and Wei Hu. Learning to exploit long-term relational dependencies in knowledge graphs. *arXiv preprint arXiv:1905.04914*, 2019.

[107] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[108] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[109] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[110] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[111] Susan Hanson and James O Huff. Assessing day-to-day variability in complex travel patterns. *Transportation Research Record*, 891:18–24, 1981.

[112] Susan Hanson and O. James Huff. Systematic variability in repetitious travel. *Transportation*, 15(1-2):111–135, 1988.

[113] Tanzima Hashem and Lars Kulik. Safeguarding Location Privacy in Wireless Ad-Hoc Networks. In *UbiComp 2007: Ubiquitous Computing*, pages 372–390. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[114] Marwan Hassani, Philipp Kranen, and Thomas Seidl. Precise anytime clustering of noisy sensor data with logarithmic complexity. In *Proceedings of the Fifth International Workshop on Knowledge Discovery from Sensor Data*, pages 52–60. ACM, 2011.

[115] Marwan Hassani, Pascal Spaus, Mohamed Gaber, and Thomas Seidl. Density-based projected clustering of data streams. *Scalable Uncertainty Management*, pages 311–324, 2012.

[116] Xiaofei He and Partha Niyogi. Locality preserving projections. In *Advances in neural information processing systems*, pages 153–160, 2004.

[117] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. Rolx: structural role extraction & mining in large graphs. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1231–1239. ACM, 2012.

[118] Paul W Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. Stochastic blockmodels: First steps. *Social networks*, 5(2):109–137, 1983.

[119] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM Journal on Computing*, 2(4):225–231, dec 1973.

[120] Kathleen Stewart Hornsby and Stephen Cole. Modeling Moving Geospatial Objects from an Event-based Perspective. *Transactions in GIS*, 11(4):555–573, aug 2007.

[121] Frank E. Horton and David R. Reynolds. Effects of Urban Spatial Structure on Individual Behavior. *Economic Geography*, 47(1):36–48, 1971.

[122] Paul VC Hough. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654.

[123] Hao Huang, Shinjae Yoo, and Shiva Prasad Kasiviswanathan. Unsupervised feature selection on data streams. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1031–1040. ACM, 2015.

[124] Qunying Huang and David W. S. Wong. Modeling and Visualizing Regular Human Mobility Patterns with Uncertainty: An Example Using Twitter Data. *Annals of the Association of American Geographers*, 105(6):1179–1197, 2015.

[125] Qunying Huang and David W. S. Wong. Activity patterns, socioeconomic status and urban spatial structure: what can social media data tell us? *International Journal of Geographical Information Science*, 8816(February):1–26, 2016.

[126] L Hubert and P Arabie. Comparing partitions journal of classification 2 193–218. *Google Scholar*, 1985.

[127] IMDb. The internet movie database. In *http://www.imdb.com/*, 2016, last accessed: 2016-11-22.

[128] Paul Jaccard. The distribution of the flora in the alphine zone. *The New Phytologist*, XI(2):37–50, 1912.

[129] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[130] Paul A Jargowsky. Ghetto poverty among blacks in the 1980s. *Journal of Policy Analysis and Management*, 13(2):288–310, 1994.

[131] Paul A Jargowsky. *Poverty and place: Ghettos, barrios, and the American city.* Russell Sage Foundation, 1997.

[132] Olle Järv, Kerli Müürisepp, Rein Ahas, Ben Derudder, and Frank Witlox. Ethnic differences in activity spaces as a characteristic of segregation: A study based on mobile phone usage in Tallinn, Estonia. *Urban Studies*, 52(14):2680–2698, nov 2015.

[133] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*, pages 271–279. ACM, 2003.

[134] Lucas GS Jeub, Prakash Balachandran, Mason A Porter, Peter J Mucha, and Michael W Mahoney. Think locally, act locally: Detection of small, medium-sized, and large communities in large networks. *Physical Review E*, 91(1):012821, 2015.

[135] Pan Ji, Tong Zhang, Hongdong Li, Mathieu Salzmann, and Ian Reid. Deep subspace clustering networks. In *Advances in Neural Information Processing Systems*, pages 24–33, 2017.

[136] Shihao Ji, Nadathur Satish, Sheng Li, and Pradeep Dubey. Parallelizing word2vec in shared and distributed memory. *arXiv preprint arXiv:1604.04661*, 2016.

[137] Karin Kailing, Hans-Peter Kriegel, and Peer Kröger. Density-connected subspace clustering for high-dimensional data. In *Proceedings of the 2004 SIAM international conference on data mining*, pages 246–256. SIAM, 2004.

[138] Daniyal Kazempour, Kevin Bein, Peer Kröger, and Thomas Seidl. D-masc: A novel search strategy for detecting regions of interest in linear parameter space. In *International Conference on Similarity Search and Applications*, pages 163–176. Springer, 2018.

[139] Daniyal Kazempour, Markus Mauder, Peer Kröger, and Thomas Seidl. Detecting global hyperparaboloid correlated clusters based on hough transform. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, page 31. ACM, 2017.

[140] Daniyal Kazempour, Markus Mauder, Peer Kröger, and Thomas Seidl. Detecting global hyperparaboloid correlated clusters: a hough-transform based multicore algorithm. *Distributed and Parallel Databases*, pages 1–34, 2018.

[141] Kristian Kersting, Nils M. Kriege, Christopher Morris, Petra Mutzel, and Marion Neumann. Benchmark data sets for graph kernels, 2016.

[142] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[143] D Knuth. The art of computer programming: Vol 2/seminumerical algorithms, chapter 3: Random numbers, 1969.

[144] Danai Koutra, Tai-You Ke, U Kang, Duen Horng Polo Chau, Hsing-Kuo Kenneth Pao, and Christos Faloutsos. Unifying guilt-by-association approaches: Theorems and fast algorithms. In *Proc. of ECML PKDD*, pages 245–260. Springer, 2011.

[145] Philipp Kranen, Ira Assent, Corinna Baldauf, and Thomas Seidl. The clustree: indexing micro-clusters for anytime stream mining. *Knowledge and information systems*, 29(2):249–272, 2011.

[146] H-P Kriegel, Peer Kroger, Matthias Renz, and Sebastian Wurst. A generic framework for efficient subspace clustering of high-dimensional

data. In *fifth IEEE international conference on data mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[147] Hans-Peter Kriegel, Peer Kröger, Irene Ntoutsi, and Arthur Zimek. Towards subspace clustering on dynamic data: an incremental version of predecon. In *Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques*, pages 31–38. ACM, 2010.

[148] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1, 2009.

[149] Quoc V Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the International conference on machine learning*, volume 14, pages 1188–1196, 2014.

[150] John Boaz Lee, Ryan Rossi, and Xiangnan Kong. Graph classification using structural attention. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1666–1674. ACM, 2018.

[151] Kristen LeFevre, D.J. DeWitt, and Raghu Ramakrishnan. Mondrian Multidimensional K-Anonymity. In *22nd International Conference on Data Engineering (ICDE'06)*, volume 2006, pages 25–25. IEEE, 2006.

[152] J. Leskovec, K.J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web*, pages 695–704, 2008.

[153] J. Leskovec, K.J. Lang, and M. W. Mahoney. Empirical comparison of algorithms for network community detection. In *Proceedings of the 19th International Conference on World Wide Web*, pages 631–640, 2010.

[154] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[155] Jure Leskovec, Kevin J Lang, Anirban Dasgupta, and Michael W Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[156] Ron Levie, Federico Monti, Xavier Bresson, and Michael M. Bronstein. Cayleynets: Graph convolutional neural networks with complex rational spectral filters. *CoRR*, abs/1705.07664, 2017.

[157] Omer Levy and Yoav Goldberg. Dependency-based word embeddings. In *ACL (2)*, pages 302–308, 2014.

[158] Cheng Li, Xiaoxiao Guo, and Qiaozhu Mei. Deepgraph: Graph structure predicts network growth. *arXiv preprint arXiv:1610.06251*, 2016.

[159] Kang Li, Jing Gao, Suxin Guo, Nan Du, Xiaoyi Li, and Aidong Zhang. Lrbm: A restricted boltzmann machine based approach for representation learning on linked data. In *Proceedings of 2014 IEEE International Conference on Data Mining*, pages 300–309. IEEE, 2014.

[160] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. t-Closeness: Privacy Beyond k-Anonymity and l-Diversity. In *2007 IEEE 23rd International Conference on Data Engineering*, number 2, pages 106–115. IEEE, 2007.

[161] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. Adaptive graph convolutional neural networks. *arXiv preprint arXiv:1801.03226*, 2018.

[162] Toby Jia-Jun Li, Shilad Sen, and Brent Hecht. Leveraging advances in natural language processing to better understand tobler's first law of geography. In *ACM SIGSPATIAL*, pages 513–516. ACM, 2014.

[163] Yongmin Li. On incremental and robust subspace learning. *Pattern recognition*, 37(7):1509–1518, 2004.

[164] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. Graph matching networks for learning the similarity of graph structured objects. *arXiv preprint arXiv:1904.12787*, 2019.

[165] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. In *arXiv preprint arXiv:1511.05493*, 2015.

[166] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *Journal of the Association for Information Science and Technology*, 58(7):1019–1031, 2007.

[167] Bing Liu, Yiyuan Xia, and Philip S Yu. Clustering through decision tree construction. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 20–29. ACM, 2000.

[168] Guimei Liu, Jinyan Li, Kelvin Sim, and Limsoon Wong. Distance based subspace clustering with flexible dimension partitioning. In *2007 IEEE 23rd International Conference on Data Engineering*, pages 1250–1254. IEEE, 2007.

[169] Guimei Liu, Kelvin Sim, Jinyan Li, and Limsoon Wong. Efficient mining of distance-based subspace clusters. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):427–444, 2009.

[170] Jing Liu, Fan Zhang, Xinying Song, Young-In Song, Chin-Yew Lin, and Hsiao-Wuen Hon. What's in a name? In *Proceedings of the sixth ACM international conference on Web search and data mining - WSDM '13*, page 495, New York, New York, USA, 2013. ACM Press.

[171] Li Liu, William K Cheung, Xin Li, and Lejian Liao. Aligning users across social networks using network embedding. In *IJCAI*, pages 1774–1780, 2016.

[172] Siyuan Liu, Shuhui Wang, Feida Zhu, Jinbo Zhang, and Ramayya Krishnan. HYDRA: large-scale social identity linkage via heterogeneous behavior modeling. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data - SIGMOD '14*, pages 51–62, New York, New York, USA, 2014. ACM Press.

[173] Tao Liu, Shengping Liu, Zheng Chen, and Wei-Ying Ma. An evaluation on feature selection for text clustering. In *ICML*, volume 3, pages 488–495, 2003.

[174] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[175] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[176] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[177] Ashwin Machanavajjhala, Johannes Gehrke, Daniel Kifer, and Muthuramakrishnan Venkitasubramaniam. L-diversity: privacy beyond k-anonymity. In *22nd International Conference on Data Engineering (ICDE'06)*, volume 2006, pages 24–24. IEEE, 2006.

[178] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[179] Anshu Malhotra, Luam Totti, Wagner Meira, Ponnurangam Kumaraguru, and Virgilio Almeida. Studying User Footprints in Different Online Social Networks. In *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1065–1070. IEEE, aug 2012.

[180] Yann LeCun Mikael Henaff, Joan Bruna. Deep convolutional networks on graph-structured data. *arXiv preprint arXiv:1506.05163*, 2015.

[181] T Mikolov and J Dean. Distributed representations of words and phrases and their compositionality. *Proceedings of Advances in neural information processing systems*, 2013.

[182] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[183] Vinith Misra and Sumit Bhatia. Bernoulli embeddings for graphs. *arXiv preprint arXiv:1803.09211*, 2018.

[184] Gabriela Moise, Jorg Sander, and Martin Ester. P3c: A robust projected clustering algorithm. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 414–425. IEEE, 2006.

[185] Gabriela Moise, Jörg Sander, and Martin Ester. Robust projected clustering. *Knowledge and Information Systems*, 14(3):273–298, 2008.

[186] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M. Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *CoRR*, abs/1611.08402, 2016.

[187] Emmanuel Müller, Ira Assent, Stephan Günnemann, Ralph Krieger, and Thomas Seidl. Relevant subspace clustering: Mining the most interesting non-redundant concepts in high dimensional data. In *2009*

Ninth IEEE International Conference on Data Mining, pages 377–386. IEEE, 2009.

[188] Emmanuel Müller, Ira Assent, Stephan Günnemann, and Thomas Seidl. Scalable density-based subspace clustering. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1077–1086. ACM, 2011.

[189] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[190] Elham Naghizade, James Bailey, Lars Kulik, and Egemen Tanin. How private can I be among public users? In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '15*, pages 1137–1141, New York, New York, USA, 2015. ACM Press.

[191] Galileo Namata, Ben London, Lise Getoor, Bert Huang, and UMD EDU. Query-driven active surveying for collective classification. In *10th International Workshop on Mining and Learning with Graphs*, 2012.

[192] Annamalai Narayanan, Mahinthan Chandramohan, Lihui Chen, Yang Liu, and Santhoshkumar Saminathan. subgraph2vec: Learning distributed representations of rooted sub-graphs from large graphs. *arXiv preprint arXiv:1606.08928*, 2016.

[193] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856, 2002.

[194] Dang Nguyen, Wei Luo, Tu Dinh Nguyen, Svetha Venkatesh, and Dinh Phung. Learning graph representation via frequent subgraphs. In *Proceedings of the 2018 SIAM International Conference on Data Mining*, pages 306–314, 2018.

[195] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *Proceedings of the 2016 International conference on machine learning*, pages 2014–2023, 2016.

[196] Giannis Nikolentzos, Polykarpos Meladianos, and Michalis Vazirgiannis. Matching node embeddings for graph similarity. In *Proceedings of*

the 31st AAAI Conference on Artificial Intelligence, pages 2429–2435, 2017.

[197] Irene Ntoutsi, Arthur Zimek, Themis Palpanas, Peer Kröger, and Hans-Peter Kriegel. Density-based projected clustering over high dimensional data streams. In *Proceedings of the 2012 SIAM international conference on data mining*, pages 987–998. SIAM, 2012.

[198] Liadan O'Callaghan, Nina Mishra, Adam Meyerson, Sudipto Guha, and Rajeev Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings 18th International Conference on Data Engineering*, pages 685–694. IEEE, 2002.

[199] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[200] Daniel W Otter, Julian R Medina, and Jugal K Kalita. A survey of the usages of deep learning in natural language processing. *arXiv preprint arXiv:1807.10854*, 2018.

[201] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[202] Nam Hun Park and Won Suk Lee. Grid-based subspace clustering over data streams. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 801–810. ACM, 2007.

[203] Vishal M Patel, Hien Van Nguyen, and René Vidal. Latent space sparse subspace clustering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 225–232, 2013.

[204] Vishal M Patel, Hien Van Nguyen, and René Vidal. Latent space sparse and low-rank subspace clustering. *IEEE Journal of Selected Topics in Signal Processing*, 9(4):691–701, 2015.

[205] Vishal M Patel and René Vidal. Kernel sparse subspace clustering. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2849–2853. IEEE, 2014.

[206] Judea Pearl. *Reverend Bayes on inference engines: A distributed hierarchical approach.* Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

[207] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine Learning in Python. In *Journal of Machine Learning Research*, volume 12, pages 2825–2830. 2012.

[208] Leto Peel. Graph-based semi-supervised learning for relational networks. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 435–443. SIAM, 2017.

[209] Leto Peel, Jean-Charles Delvenne, and Renaud Lambiotte. Multiscale mixing patterns in networks. *Proceedings of the National Academy of Sciences*, 115(16):4057–4062, 2018.

[210] Shichao Pei, Lu Yu, Robert Hoehndorf, and Xiangliang Zhang. Semi-supervised entity alignment via knowledge graph embedding with awareness of degree difference. In *The World Wide Web Conference*, pages 3130–3136. ACM, 2019.

[211] Xi Peng, Shijie Xiao, Jiashi Feng, Wei-Yun Yau, and Zhang Yi. Deep subspace clustering with sparsity prior. In *IJCAI*, pages 1925–1931, 2016.

[212] Xi Peng, Zhang Yi, and Huajin Tang. Robust subspace clustering via thresholding ridge regression. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[213] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[214] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710, 2014.

[215] Cecilia M Procopiuc, Michael Jones, Pankaj K Agarwal, and TM Murali. A monte carlo algorithm for fast projective clustering. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 418–427. ACM, 2002.

[216] William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

[217] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394, 2017.

[218] Azriel Rosenfeld. Picture processing by computer. volume 1, pages 147–176. ACM, 1969.

[219] Ryan A Rossi and Nesreen K Ahmed. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1112–1131, 2015.

[220] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[221] Juan J Ruiz, F Javier Ariza, Manuel A Urena, and Elidia B Blázquez. Digital map conflation: a review of the process and a proposal for classification. *International Journal of Geographical Information Science*, 25(9):1439–1466, 2011.

[222] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, jan 1988.

[223] Alberto Sanfeliu and King-Sun Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE transactions on systems, man, and cybernetics*, (3):353–362, 1983.

[224] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2008.

[225] Satu Elisa Schaeffer. Graph clustering. *Computer science review*, 1(1):27–64, 2007.

[226] Harald Scheid and Wolfgang Schwarz. *Elemente der linearen Algebra und der Analysis*. Springer-Verlag, 2009.

[227] Erich Schubert, Michael Weiler, and Hans-Peter Kriegel. Signitrend: scalable detection of emerging topics in textual streams by hashed significance thresholds. In *KDD*, pages 871–880. ACM, 2014.

[228] Erik Seglem, Andreas Züfle, Jan Stutzki, Felix Borutta, Evgheniy Faerman, and Matthias Schubert. On privacy in spatio-temporal data: User identification using microblog data. In *International Symposium on Spatial and Temporal Databases*, pages 43–61. Springer, 2017.

[229] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93, 2008.

[230] Yue Shen, Mei-Po Kwan, and Yanwei Chai. Investigating commuting flexibility with GPS data and 3D geovisualization: a case study of Beijing, China. *Journal of Transport Geography*, 32:1–11, oct 2013.

[231] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[232] Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial Intelligence and Statistics*, pages 488–495, 2009.

[233] Julian Shun and Guy E Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *ACM SIGPLAN Notices*, volume 48, pages 135–146. ACM, 2013.

[234] Julian Shun, Farbod Roosta-Khorasani, Kimon Fountoulakis, and Michael W. Mahoney. Parallel local graph clustering. *Proceedings of the VLDB Endowment*, 9(12):1041–1052, August 2016.

[235] Jonathan A Silva, Elaine R Faria, Rodrigo C Barros, Eduardo R Hruschka, Andre CPLF De Carvalho, and João Gama. Data stream clustering: A survey. *ACM Computing Surveys (CSUR)*, 46(1):13, 2013.

[236] Martin Simonovsky and Nikos Komodakis. Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[237] Josef Sivic and Andrew Zisserman. Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence*, 31(4):591–606, 2009.

[238] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[239] Anthony Stefanidis, Andrew Crooks, and Jacek Radzikowski. Harvesting ambient geospatial information from social media feeds. *GeoJournal*, 78(2):319–338, apr 2013.

[240] Zequn Sun, Wei Hu, Qingheng Zhang, and Yuzhong Qu. Bootstrapping entity alignment with knowledge graph embedding. In *IJCAI*, pages 4396–4402, 2018.

[241] Aram Arutiunovich Sveshnikov and Bernard R Gelbaum. *Problems in probability theory, mathematical statistics and theory of random functions*. Courier Corporation, 1968.

[242] Latanya Sweeney. k-ANONYMITY: A MODEL FOR PROTECTING PRIVACY. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, oct 2002.

[243] Amos Tanay, Roded Sharan, and Ron Shamir. Biclustering algorithms: A survey. *Handbook of computational molecular biology*, 9(1-20):122–124, 2006.

[244] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th international conference on world wide web*, pages 1067–1077. ACM, 2015.

[245] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.*

[246] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[247] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. Learning deep representations for graph clustering. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1293–1299, 2014.

[248] Waldo R Tobler. A computer movie simulating urban growth in the detroit region. *Economic geography*, 46(sup1):234–240, 1970.

[249] Warren S Torgerson. Theory and methods of scaling. 1958.

[250] Bayu Distiawan Trisedya, Jianzhong Qi, and Rui Zhang. Entity alignment between knowledge graphs using attribute embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 297–304, 2019.

[251] Rolando Trujillo-Rasua and Josep Domingo-Ferrer. On the privacy offered by (k, δ)-anonymity. *Information Systems*, 38(4):491–494, jun 2013.

[252] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, Alex Bronstein, and Emmanuel Müller. Netlsd: Hearing the shape of a graph. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018.

[253] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. Verse: Versatile graph embeddings from similarity measures. In *Proc. of WWW*, pages 539–548, 2018.

[254] Ke Tu, Peng Cui, Xiao Wang, Philip S Yu, and Wenwu Zhu. Deep recursive network embedding with regular equivalence. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2357–2366. ACM, 2018.

[255] Anthony KH Tung, Xin Xu, and Beng Chin Ooi. Curler: finding and visualizing nonlinear correlation clusters. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 467–478, 2005.

[256] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[257] Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

[258] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

[259] Bo Wang, Zhuowen Tu, and John K Tsotsos. Dynamic label propagation for semi-supervised multi-class multi-label classification. In

Proceedings of the IEEE international conference on computer vision, pages 425–432, 2013.

[260] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234. ACM, 2016.

[261] Zhichun Wang, Qingsong Lv, Xiaohan Lan, and Yu Zhang. Cross-lingual knowledge graph alignment via graph convolutional networks. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 349–357, 2018.

[262] Michael Weiler, Andreas Züfle, Felix Borutta, and Tobias Emrich. Socio textual mapping. In *Proceedings of the 8th ACM SIGSPATIAL International Workshop on Location-Based Social Networks*, page 6. ACM, 2015.

[263] Jason Weston, Frédéric Ratle, Hossein Mobahi, and Ronan Collobert. Deep learning via semi-supervised embedding. In *Neural Networks: Tricks of the Trade*, pages 639–655. Springer, 2012.

[264] David W S Wong and Shih-Lung Shaw. Measuring segregation: an activity space approach. *Journal of Geographical Systems*, 13(2):127–145, jun 2011.

[265] Kyoung-Gu Woo, Jeong-Hoon Lee, Myoung-Ho Kim, and Yoon-Joon Lee. Findit: a fast and intelligent subspace clustering algorithm using dimension voting. *Information and Software Technology*, 46(4):255–271, 2004.

[266] Kun Xu, Liwei Wang, Mo Yu, Yansong Feng, Yan Song, Zhiguo Wang, and Dong Yu. Cross-lingual knowledge graph alignment via graph matching neural network. *arXiv preprint arXiv:1905.11605*, 2019.

[267] Wei Xu, Xin Liu, and Yihong Gong. Document clustering based on non-negative matrix factorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 267–273. ACM, 2003.

[268] Shuicheng Yan, Dong Xu, Benyu Zhang, and Hong-Jiang Zhang. Graph embedding: A general framework for dimensionality reduction. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 830–837. IEEE, 2005.

[269] Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1365–1374. ACM, 2015.

[270] Bisheng Yang, Yunfei Zhang, and Xuechen Luan. A probabilistic relaxation approach for matching road networks. *International Journal of Geographical Information Science*, 27(2):319–338, 2013.

[271] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd International Conference on Machine Learning*, pages 40–48, 2016.

[272] Kevin Y Yip, David W Cheung, and Michael K Ng. Harp: A practical projected clustering algorithm. *IEEE Transactions on knowledge and data engineering*, 16(11):1387–1397, 2004.

[273] Kevin Y Yip, David W Cheung, and Michael K Ng. On discovery of extremely low-dimensional clusters using semi-supervised projected clustering. In *Proceedings-International Conference On Data Engineering*. IEEE, Computer Society., 2005.

[274] Man Lung Yiu and Nikos Mamoulis. Frequent-pattern based iterative projected clustering. In *Third IEEE International Conference on Data Mining*, pages 689–692. IEEE, 2003.

[275] Man Lung Yiu and Nikos Mamoulis. Iterative projected clustering by subspace mining. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):176–189, 2005.

[276] Reza Zafarani and Huan Liu. Connecting Corresponding Identities across Communities. *Proceedings of the Third International Conference on Weblogs and Social Media - ICWSM 2009*, 9(November):354–357, 2009.

[277] Qingheng Zhang, Zequn Sun, Wei Hu, Muhao Chen, Lingbing Guo, and Yuzhong Qu. Multi-view knowledge graph embedding for entity alignment. *arXiv preprint arXiv:1906.02390*, 2019.

[278] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: an efficient data clustering method for very large databases. In *ACM Sigmod Record*, volume 25, pages 103–114. ACM, 1996.

[279] Ziwei Zhang, Peng Cui, Xiao Wang, Jian Pei, Xuanrong Yao, and Wenwu Zhu. Arbitrary-order proximity preserved network embedding. In *Proc. of SIGKDD*, pages 2778–2786, 2018.

[280] Aoying Zhou, Feng Cao, Weining Qian, and Cheqing Jin. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems*, 15(2):181–214, 2008.

[281] Denny Zhou, Olivier Bousquet, Thomas N Lal, Jason Weston, and Bernhard Schölkopf. Learning with local and global consistency. In *Proceedings of Advances in neural information processing systems*, pages 321–328, 2004.

[282] Pan Zhou, Yunqing Hou, and Jiashi Feng. Deep adversarial subspace clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1596–1604, 2018.

[283] Qiannan Zhu, Xiaofei Zhou, Jia Wu, Jianlong Tan, and Li Guo. Neighborhood-aware attentional representation for multilingual knowledge graphs. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 1943–1949. International Joint Conferences on Artificial Intelligence Organization, 2019.

[284] Xiaojin Zhu and Zoubin Ghahramani. Learning from labeled and unlabeled data with label propagation. 2002.

[285] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003.

[286] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.