Representation learning of user-generated data

Mickael Poussevin

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité
Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Mickaël Poussevin

Pour obtenir le grade de
DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Apprentissage de representation pour des données générées par des utilisateurs

***

Representation Learning Of User-generated Data

réalisée à:
THALES COMMUNICATIONS & SECURITY et LABORATOIRE D'INFORMATIQUE DE PARIS 6

soutenue le 21 janvier 2014
devant le jury composé de :

M. Massih-Reza AMINI	Rapporteur
M. Emmanuel VIENNET	Rapporteur
M. Catherine GOUTTAS	Examinateur
M. Bernd AMANN	Examinateur
M. Patrick GALLINARI	Directeur de thèse
M. Vincent GUIGUE	Encadrant de thèse

Acknowledgement

Contents

Abstract

Computer science currently undergoes a profound mutation. Gains in computational power associated to the availability of large datasets provide a fertile ground for the development of machine learning research. First, it is now possible to train complex representation learning models such as deep neural networks or advanced matrix factorizations that learn optimal representation spaces. The long term goal behind representation learning is the design of completely automated machine learning based processing chains, from raw input to final outputs. Such complex models are prone to over-fitting. They require intensive computations and large datasets. They use strong complexity control mechanisms as well as model selection techniques to obtain sensible solutions. Second, this mutation opens new opportunities by providing large corpora of data generated by users that requires new types of analyses and provides new applications. The main challenge in this context is to take into account user preferences to personalize information access in the digital world. Machine learning algorithms already are ubiquitous: personalized answers in search engines, item recommendations and personalized ads are now part of the Internet landscape. It drives machine learning research towards another of its long term goals: the design of context aware models.

In this thesis, we study how representation learning methods can be applied to user-generated data. Our contributions cover three different applications but share a common denominator: the extraction of relevant user representations. Our first application is the item recommendation task, where recommender systems build user and item profiles out of past ratings reflecting user preferences and item characteristics. Nowadays, textual information is often together with ratings available and we propose to use it to enrich the profiles extracted from the ratings. Our hope is to extract from the textual content shared opinions and preferences. The models we propose provide another opportunity: predicting the text a user would write on an item. We formalize this task as the generation of a synthetic review made of relevant sentences selected among the sentences written by other users. To the best of our knowledge we are the first to consider such a task to exploit the textual information of user review with recommender systems as well as to personalized even more the item recommendations made to users by presenting them additional information. Our second application is sentiment analysis and, in particular, polarity classification. For the latter, the goal is to design generic models that decide whether a given text expresses a positive or negative opinion. The domain largely benefited from the availability of user reviews, as did recommender systems, since they provide large supervised datasets to train such models. Our idea is that recommender systems can be used for such a task. Recommender systems and traditional polarity classifiers operate on different time scales. We propose two hybridizations of these models: the former has better classification performance, the latter highlights a vocabulary of surprise in the texts of the reviews. The third and final application we consider is urban mobility. It takes place beyond the frontiers of the Internet, in the physical world. Using authentication logs of the subway users, logging the time and station at which users take the subway, we show that it is possible to extract robust temporal profiles. We provide a matrix factorization algorithm that learns an easily interpretable representations of users based on activities. Our matrix factorization enables us to learn both user profiles and activities at the same time.

1 Introduction

Contents

Information systems and computer science currently undergo a profound mutation as shown by the *Big Data* phenomenon. This mutation takes root jointly in the growth in computing power and storage capacities of IT infrastructures and in the advances of data acquisition techniques. It provides a fertile ground for the rapid development of research in machine learning, providing the appropriate context to train complex models such as advanced matrix factorization or deep neural networks, which involve large numbers of free parameters. First, recent IT infrastructures are powerful enough to handle the intensive computation necessary to solve the difficult optimization problems of training such models while using complexity control mechanisms and model selection techniques. Second, datasets are now larger and larger. For example, in image classification tasks, the old MNIST dataset [LC+90] contains 60k training examples (handwritten digits) and 10 classes while the dataset of the ImageNet classification challenge [Rus+14] contains 500k examples and 1000 classes. Similarly, for natural language processing and sentiment analysis, huge textual datasets are available as shown by the dumps of Wikipedia exploited in [Mik+13] and by the large quantities of user reviews [JL08]. Using more training example is in itself a complexity control mechanism, the sample of observed data is more exhaustive, which limits the over-fitting is limited.

For machine learning research, the first opportunity provided by this changing context is the opportunity to extend traditional analyses and models. This is reflected, in particular, by the rapid development of representation learning methods which we study throughout this thesis. Representation learning aims at replacing the tedious task of hand-crafting representations of the data by the design of completely automated machine learning based processing chains, a long term goal of artificial intelligence. The performance of machine learning algorithms is strongly impacted by the quality of the representation, discriminant features drastically helping the models. The features where traditionally hand-crafted by analysts after reviewing the data and task at hand. This generally is a long and costly process. For instance, convolutional neural networks [Law+97] can take advantage of the large datasets of images available on the Internet and extract automatically pertinent descriptors that provide better classification performances than traditional SIFT descriptors [Kri+12]. For other signal processing tasks such as handwriting recognition and speech recognition, recurrent neural networks [Dor96] are now state of the art models [Gra12] that build context dependent representations.

The second opportunity for the enrichment the field is the availability of new types of data and of new applications. This is particularly true for the analysis and modeling of user behaviors and preferences, a new field, as described in [FH10], to which belong the models and analyses provided in this thesis. Considering the user is the first step toward another long term goal of machine learning: the ability to consider contexts. Even though the domain is relatively recent, it is already ubiquitous and many Internet dwellers use such models on a daily basis. Search engines analyze the browsing history of users to re-rank results of queries accordingly to the preferences of the users [Tee+05] User profiles are exploited in many digital applications, due to the easiness of the data collection, such as on-line dating [Dia+10], item recommendation of merchant website [Kan+11] or bidding on advertising spaces on the Internet [Cha+10]. While considering user preferences is still absent from many domains, such as sentiment analysis [PL08], the approach tends to be more and more widely adopted, even beyond the frontiers of the digital world, for instance in urban mobility studies where the understanding of user behavior is a major issue for the development and maintenance of large urban networks.

This thesis is at the interplay of both representation learning and user behavior analysis It studies the application of representation learning methods for the extraction of robust, informative and interpretable user profiles. This introduction presents the challenges and context of the two axes, starting with representation learning and continuing with the extraction of user profiles.

1.1 Representation learning

As said before, research in representation learning is driven by the long term goal of designing fully automated machine learning based processing chains, from the data acquisition to the final output such as object recognition in a stream of images. The interest of using representation learning methods is twofold: to replace the tedious task of hand-crafting features for large and complex datasets such as images or natural texts and to benefit from massive datasets to extract robust representations. Among the many approaches proposed in the literature, we focus, in this thesis, on two families of models: artificial neural networks and matrix factorization. These families compute the new representation of the data in different ways. The former learn coding functions, often associated with an inverse decoding function, to project data to into a new representation space. The latter extend classic bag of word representations, typical for image, text and signal applications, where a dictionary of basic elements is used to compute the representation, by learning simultaneously the dictionary and the optimal representation of data with respect to the dictionary.

As we will see in details in chapter 2, neural networks are representation learning method based on the principle of coding functions. Many different formalisms have been proposed for neural networks, such as the probabilistic Boltzmann Machines and their restricted variants [HS06]. In this thesis, we focus on the deterministic formalism of multi-layer perceptrons [LC+90], as it is widely used today for representation learning applied to natural language processing and sentiment analysis [Col+11; Mik+13; LM14]. Within this formalism, each layer of a multi-layer perceptron learns a coding or encoding function. In the typical setting, a MLP is composed of one hidden layer and one output layer. Optimizing a loss function, the design and training of neural networks, through back-propagation for example, imposes that the coding function learned by the hidden layer disentangles the data, to allow for good performance. However, when multiple hidden layers are used, the gradient is rapidly diluted and only impact layers close to the output, a phenomenon called the vanishing of the gradient. A solution, which allows to learn deep neural networks [Ben+07], is to train each layer successively as hidden layers of auto-encoders. These models, also known as self-associative memories [Sou+87], are composed of one hidden and one output layer. The former learns a coding function and the latter the inverse decoding function. They are trained in an unsupervised way to correctly reconstruct data and this necessity of correct reconstruction replaces the necessity of correct classification of traditional MLPs and imposes that the hidden layer learns disentangled representations as well. Taking advantage of the theoretical knowledge gained on connectionist models [Bot91] and of recent gains in computational power, neural networks have imposed themselves as state of the art models for many signal processing tasks as object recognition in images [Cha+14], handwriting and speech recognition [Gra12] and natural language processing [Col+11; Mik+13; LM14]. However, explicating the decisions of the (highly non-linear) coding functions learned by neural networks remains very difficult, even counterintuitive [Sze+13]. Neural networks are and remain black boxes.

Matrix factorization [LS01; Mai+10; GVL12] offers an interesting alternative to neural networks. They do not learn parametric coding functions. Matrix factorization, studied in chapter 3, proposes to learn a set of basic patterns from the data, gathered in a dictionary, and to represent data in the space defined by these patterns, at the same time. For signal processing applications, this means that the signal is encoded using a basis of vector corresponding to a dictionary that is learned at same time as the code. This is a difficult optimization problem and, like neural networks, it requires complexity control mechanisms to extract sensible representations. As neural networks, matrix factorization benefits from the gains in computational power [Yin+14] since they can be efficiently parallelized and can then be trained on large datasets. Control mechanisms include constraining the rank of the terms involved for SVD (Singular Value Decomposition) [GVL12], imposing that elements of the dictionary are uncorrelated like for PCA (Principal Component Analysis) [Pea01; Hot33] or independent like for ICA (Independent Component Analysis) [Com94]. Many variations have been proposed such as non-negative matrix factorization [LS01] or sparse matrix factorization [Hoy02; Hoy04] to adapt the factors to the nature of data. The drawback of matrix factorization, known at the out-of-sample problem, is the absence of coding function to easily compute the representation of new examples. Computing the code of new examples can however be done efficient while updating the dictionary at the same time, such as [BG09] does to remove the background of video streams. Their advantage is to provide easily interpretable representations, in particular with the use of non-negativity constraints [LS99].

1.2 User profiles

One of the deepest impact of the *Big Data* mutation is the availability of large dataset of user traces that led to a proliferation of studies on user behaviors. The long term goal behind this issue is to design context aware machine learning models: considering user preferences and/or browsing history amounts at creating a context to provide a personalized access to information. While the analysis of user preferences is a recent field [FH10], it is already ubiquitous on the Internet and many people use such models transparently on a daily basis: recommender systems and customized advertising are part of the Internet landscape. It is a field of intensive research, driven by major economical issues, as revealed by the 1M$ prize awarded to the winner of the Netflix challenge [BL07]. For information retrieval, one of the most common applications is the ranking of query results based on the preferences of users as most search engines do [Tee+05]. Similar problems arise when searching for compatible profiles on online dating sites [ML10; Dia+10] Finally, recommender systems, famous in the community since the Netflix challenge [BL07], use the navigation history of users to build profiles representative of their taste and to recommend pertinent items.

Our first contributions in chapters 4 and 5 concern item recommendation. Among the various approaches proposed to address this task, we have chosen the collaborative filtering framework [KB11] which exploits the large quantities of reviews available when content based approaches rely on accurate description of items that are often missing. These methods are typically based on matrix factorization [Kor08; Kor+09], using the ratings given by users on items. Matrix factorization provides reference models for blind source separation problems and recommender systems are an example of such a problem on dyadic data. Dyadic data are the result of the interaction of two types of entities, such as transactions between a credit card holder and a merchant or a review a user writes about an item. Recommender systems can be interpreted as explaining an observed signal, the ratings, by the interaction of a predetermined number of hidden sources describing user preferences and item characteristics. The challenge is to exploit as best as possible the reviews to learn profiles representative of user preferences. Our first contribution is to consider, in addition to the ratings, the texts written by users as most user reviews now contain both a text and a rating. As [ML13a] and [Gan+09], we assume that texts carries important information about user preferences and individual style that can enrich the profiles extracted by traditional recommender systems. We will consider the extraction of textual profiles along with the extraction of the traditional rating-based profiles. Our second contribution is to use these textual profiles to predict the text a user will write about an item, in addition to the rating he will give to this item. To our knowledge, we are the first to propose such a task along with an evaluation method based on summary measures [AU07]. Our models can be used to provide more personalized recommendations to users using the texts of other reviews. We show in chapter 5 that recommender systems can also be used in the context of polarity classification. We first compare various recommender systems together and then show that it is possible to use the user and item profiles that recommender systems extract to contextualize a polarity classification model [Pan+02] and improve it. Also in chapter 5, we show that it is possible to use a combination of recommender systems and polarity classification models to extract vocabulary of surprise.

Finally, among the many data generated by the current evolutions of information systems, users reviews are just one example. Traces of access to different services are another. The challenge here is to extract interpretable profiles characterizing the temporal behavior of users. This study does not take place in the digital world but in the physical world, in the context of urban mobility: we consider subway trips, as detailed in chapter 6. In this context, extracting the temporal patterns explaining the trips of individual user is important. We propose an adapted non-negative matrix factorization that extracts out of these logs a set of activities, defined as event occurring at a certain time of the day with possible repetitions during the week, and represent users in this activity space. Our study reveals links between socio-demographic patterns and the time at which users take the subway.

1.3 Thesis outline

This thesis is focused on the extraction of relevant, robust representations characterizing users from data they generate. We begin by studying two families of representation learning models: neural networks and matrix factorization. We will then present our contributions in three chapters, each chapter studies a specific application but all share a common denominator: the extraction of relevant user representations.

Chapter 2 This chapter provides background material on neural networks. We will present the deterministic formalism of multi-layer perceptrons, trained using the back-propagation algorithm. We follow a constructivist approach, starting with the neuron and assembling then in layers to build MLPs. Next, we present the auto-encoders, trained in an unsupervised fashion and used to build deep neural networks. Finally, we will present different architectures, all deterministic, such as recurrent networks and look-up tables networks.

Chapter 3 This chapter presents matrix factorization used throughout this thesis. Matrix factorization provides an alternative to neural networks, simultaneously learning an optimal dictionary to decompose data onto and the optimal decomposition. We formalize the matrix factorization as a regularized cost minimization problem through the example of the low rank approximation. Then we will present in detail two constraints commonly used: non-negativity and sparseness. We will then review various extensions of these models.

Chapter 4 This chapter presents our central contribution and takes place in the context of recommender systems. We enrich the profiles extracted on ratings, using the collaborative filtering framework, with textual profiles extracted on the texts of the user reviews. Our hypothesis is that text being a powerful media for the transmission of opinions, it will allow us to improve the characterization of users and the relevance of the recommendations. Also, considering the text allows us to use recommender systems to predict the text of a user review, in addition to its rating, as a short summary extracted out of the reviews of other users on the same item by selecting sentences that correspond to the taste and writing style of the user.

Chapter 5 In this chapter we use recommender systems in the context of polarity classification. At first, we use the classification error of our models as polarity classification models as another metric to confirm the interest of enriching rating-based profiles with textual profiles. We then show that using recommender systems to contextualize a polarity classification model improves the classification performance of the latter. In addition, we show that it is possible, by combining these models to extract a surprise vocabulary used to express a difference between the expectations of a user and reality.

Chapter 6 The two preceding chapters proceed on the same type of data generated by users, user reviews. This last chapter is concerned with authentication traces generated by the access of a user to a service. We show that it is possible to use them to extract a set of temporal patterns shared by users, and to represent users in this space. We propose an application in the context of urban mobility, extracting temporal patterns using an adapted non-negative matrix factorization. We show experimentally that temporal patterns extracted are strong characterizations of users, linked to sociological and geographical phenomena.

2 Learning representation with artificial neural networks

Contents

During their evolution, neural networks have turned to models of increasing complexity. The complexity of a statistical model can partly be quantified using the concepts of bias and variance. Given a model family and a problem, the bias measures how far the prediction functions that can be learned are from the actual function to learn. The variance measures the impact of small disturbances on the training examples on learned prediction functions. A perfect model has zero bias and low variance: it always converges to the optimal solution, but such models do not exist. Decreasing the bias is done by adding to the model degrees of freedom, for example by adding free parameters to determine during the training phase. These degrees of freedom then make the model more sensitive to variations in the training examples, thus increasing the variance. The perceptron [Ros58], which offers a simple linear decision and a robust learning algorithm, has a strong bias and low variance.

A major gain in complexity performed with neural networks has been achieved during the 80s [Bot91]. Benefiting from hardware developments and associated computational power gains, the back-propagation algorithm [Sou+87; Rum+88; BLC88] then allowed to train neural networks with enough units to deal with real problems [Bot91] such as the recognition of handwritten numbers [LC+90] used to read postal codes on envelopes. The archetype of these models is the multi-layer perceptron (MLP), a model that assembles neurons in layers and connects the layers together without loops. The back-propagation algorithm, inspired by the optimal control theory [BH69] is an iterative algorithm that propagates the errors from the output of the network back to its input. It is used to learn the parameters of MLPs, minimizing iteratively a cost function that measures the quality of the model with respect to the target task (for instance, classification in [LC+90]). In the 80s, the MLPs used in practice mainly had one hidden layer and one output layer. These MLPs have more parameters than the simple perceptrons and can learn more complex decision functions (lower bias) at the price of a stronger variance. The low bias comes from the capacity of the hidden layer to learn disentangling representations of the data that separate examples of each class, in the case of a classification task, and allows the output layer, typically linear, to correctly classifies the data. This ability to learn representations can also be exploited by training the MLPs in an unsupervised way [Sou+87]: they are trained to output a good reconstruction of their inputs. They are then called auto-associative memories or, more recently, auto-encoders. The variance of MLPs must be addressed by complexity control mechanisms, for example by using a regularization framework on the parameters, which, coupled with the strong non-convexity of the optimization problem defined by the training loss of the MLPs, makes relatively complex in practice the implementation of the back-propagation algorithm. This difficulty has been widely studied and books like [Mon+12] gather knowledge on the subject to help choosing, depending on the application, the best fitted implementation strategy. It nevertheless means that, in practice, a large number of iterations and training several models before selecting the best one is often required, which is costly in terms of computation.

The second gain in complexity is currently happening and has begun in the last decade and is characterized by the emergence of effective architectures with more than one hidden layer, called deep networks. It is based jointly on the knowledge of the 80s (back-propagation [Rum+88; BLC88] and unsupervised learning of MLPs [Sou+87]), on the newly available computing power (computer clusters [Dea+12] and graphics processing unit computing [Coa+13]) and on the availability of large amounts of data. Large datasets are still largely unsupervised as labeling data requires tedious human labor. For typical classification tasks, deep networks are pre-trained on large unlabeled datasets, then the parameters are fine-tuned for the task in question, using the labeled data. Two main families of models are used to build deep networks: Boltzmann Machines and deep MLPs. The former derive from a Bayesian formalism and is mostly used in a restricted form called Restricted Boltzmann Machines [HS06]. The unsupervised pre-training phase is performed using contrastive divergence [CPH05]. The latter are deterministic and are the natural extension of MLPs and auto-encoders, using more layers. These deterministic models are today dominant in the literature and are trained with the back-propagation algorithm [Vin+08]. In both cases, parameters are fine-tuned on labeled data using back-propagation. The argument given by proponents of deep networks [HS06; Ben+07; Hin+12; Dea+12; Glo+11; Coa+13] is that these models use the same mechanism as the MLPs, each layer learns a better representation and provides, in the end, excellent performance. These deep models have a very low bias but an unusually large variance. This variance means that today it is difficult to learn a deep network directly and the strategy developed in the 2000s to work around this problem is to train each layer of the deep model in a greedy way, one after the other [HS06; Ben+07]. Other families, such as recurrent networks [Gra12], can also build deep architectures and suffer as well from the variance problem. However, using large amounts of data and appropriate complexity of control mechanisms, neural networks have emerged as state of the art models in many areas such as image classification [HS06], voice recognition [Hin+12], handwriting recognition [Gra12]. However, if classification models learned on representations extracted by auto-encoders are effective [RS08; Cha+14], understanding the representations is difficult: the decision process of neural networks is a black box, especially for deep non-linear networks, as shown in [Sze+13].

Neural networks are now used as provide representation learning methods for text applications. Text has always been fundamental to the transmission of information over time and still is with the Internet. Text is unstructured, noisy as it contains errors (spelling, grammar) and many individual variations representative of the style of the writers. These characteristics induce many variations which can however be learned by using large text corpora. Thus, new architectures have been proposed [Col+11; Glo+11; Mik+13; LM14] to learn representations adapted to natural language processing and to text analysis.

In this chapter, we will cover the major aspects of neural networks in the context of representation learning, of which they are now one of the main family. Among the many formalisms available, we limit ourselves to the study of deterministic formalism of MLPs and auto-encoders, with back-propagation as training algorithm, as it is the most widely used in the literature on learning representations for text [RS08; Glo+11; Col+11; Mik+13; LM14]. We will present this formalism starting with the neuron and the MLPs, in the context of supervised learning. We will use these definitions to introduce representation learning with auto-encoders, which uses unsupervised learning, but are traditionally used as a brick for other classification models [RS08] or for the initialization of deep networks [Ben+07]. Finally, we will review some other deterministic architectures, such as recurrent networks [Gra12] or networks with lookup tables [Col+11; Mik+13].

2.1 Feed-forward artificial neural networks

Feed-forward neural networks are architectures without any loop: the input flows directly forward to the output. Their basic building brick is a simple computing unit (sec. 2.1.1) that mimics the behavior of neurons, of which they inherited the name. These neurons can be assembled in layers and layers may be connected together (sec. 2.1.3). Training the model is done by minimizing a loss function (sec. 2.1.2). In this thesis, we choose to minimize such function using gradient descent algorithms (sec. 2.1.2). Depending on the formalism, training neural networks can be done using first or second order methods [Bat92], contrastive divergence [CPH05] or even genetic algorithms [Bel+90]. However, steepest descent provides simple yet powerful algorithms that can be straightforwardly adapted to additional constraints on the parameters of the network – for instance for regularization purposes (sec. 2.2.2). Exploiting the absence of loops in the architecture, back-propagation proposes a clever use of the derivation chain-rule (sec. 2.1.3). This section will define all these concepts to provide the necessary background material on neural networks along with notations used throughout the thesis.

2.1.1 Perceptron

A neuron is a computing unit that – artificially – mimics the behavior of actual neurons. It takes as inputs n-dimensional vectors that we denote $\mathbf{x} \in \mathbb{R}^n$. We will consider here dot product neurons of which the output is a real number: the weighted sum of the input entries plus a bias term passed through an activation function $f : \mathbb{R} \to \mathbb{R}$. The n weights of the weighted sum are stored in the weight vector that we denote $\mathbf{w} \in \mathbb{R}^n$. We denote $b \in \mathbb{R}$ the bias or intercept term and $\theta = (\mathbf{w}, b)$ the parameters of a unit. The neuron output $g_\theta(\mathbf{x})$ is computed as:

$$g_\theta(\mathbf{x}) = f(\mathbf{w}.\mathbf{x} + b) \tag{2.1}$$

Figure 2.1 presents a neuron. Throughout the thesis, neural networks will be represented using the same formalism. Vectors are represented by successions of rectangles. Connections are represented by arrows linking rectangles. Outputs such as $g_\theta(\mathbf{x})$ (equation (2.1)) are represented by dashed arrows.

The perceptron is a neuron with f chosen as the sign function:

$$\forall r \in \mathbb{R}, f(r) = \begin{cases} 1 & \text{if } r \geq 0, \\ -1 & \text{if } r < 0. \end{cases} \tag{2.2}$$

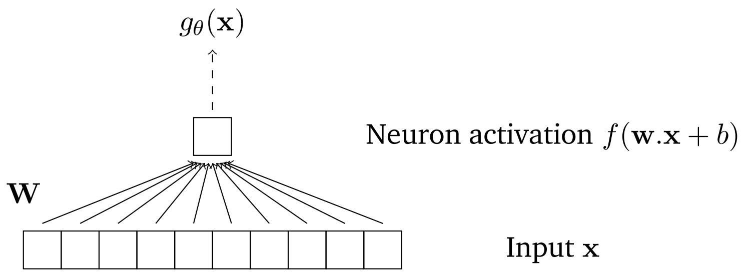


Fig. 2.1.: Representation of a neuron. From top to bottom the output, the neuron activation and the input $\mathbf{x}$.

Its learning algorithm is based on trial and error [Ros58] and is presented in algorithm 1. We present this algorithm in the context of binary classification where the goal is to correctly predict the binary label $y \in \{-1, 1\}$ of an input $\mathbf{x}$. The trial step consists in randomly selecting a random example $\mathbf{x}$ and its associated label y, compute a prediction $\hat{y}$. The error step corrects the weights of the unit if the model is mistaken. Algorithm 1 is a stochastic gradient descent that minimizes a function called the classification error.

```
Data: $\{(\mathbf{x}, y)\}, \theta_0 = (\mathbf{w}_0, b_0), \gamma, p$
Result: $\theta = (\mathbf{w}, b)$
$\mathbf{w} \leftarrow \mathbf{w}_0$, $b \leftarrow b_0$;
for $epoch \in \{1, 2, \dots, n_{epochs}\}$ do
    Randomly shuffle the training set;
    for $i \in \{1, 2, \dots, m\}$ do
        Draw the next training example $(\mathbf{x}, y)$;
        Compute $g_\theta(\mathbf{x})$;
        if $y * g_\theta(\mathbf{x}) < 0$ then
            Correct the weights $\mathbf{w} \leftarrow \mathbf{w} + \gamma y \mathbf{x}$;
            Correct the intercept $b \leftarrow b + \gamma y$;
        end
    end
end
```

Algorithm 1: The perceptron algorithm is a *trial and error* approach to learn the weights and intercept $\mathbf{w}$ and b of the neuron. During the n_{epochs} epochs, all m examples are seen in a random order and the model is updated every time a mistake is made.

2.1.2 Minimizing a loss function

We briefly present here the technique of gradient descent that we will use repeatedly throughout the thesis. This optimization technique is used to minimize functions. In the context of machine learning, the quality of models is often quantified using loss functions, that we will present next. These losses depend on the parameters of the model and training a model amounts at finding the parameters minimizing the loss.

Loss functions

The perceptron algorithm (algorithm 1) tries to minimize the number of classification errors on the training set. This number can be written as a function of the parameters θ of the model, here a neuron (weights $\mathbf{w}$ and intercept b). We denote $\mathcal{L}_{CE}$ this function:

$$\mathcal{L}_{CE}(\theta) = \mathcal{L}_{CE}(\mathbf{w}, b) = \sum_{i=1}^{m} \delta(y_i, g_\theta(\mathbf{x}_i))) \tag{2.3}$$

δ is an indicator function that equals 1 if both its parameters are equal and 0 otherwise. That is it equals 1 when y_i equals $g_\theta(\mathbf{x})$ – and 0 otherwise.

Such a function $\mathcal{L}$ is called a loss and quantifies the price of making a prediction error. In machine learning, training the model means finding the parameters such that the loss is the lowest possible. Many losses have been introduced to suit different needs. Another common one is the Mean Squared Error (MSE). We will repeatedly use it throughout the thesis. It computes the average of the squared distance between predictions $g_\theta(\mathbf{x}_i)$ and expected values y_i:

$$\mathcal{L}_{MSE}(\theta) = \frac{1}{m} \sum_{i=1}^{m} (y_i - g_\theta(\mathbf{x}_i))^2 \tag{2.4}$$

Figure 2.2 represents different loss functions in the context of binary classification. The objective is to minimize the hard zero-one loss, that is the classification error. However, the derivative of this loss is always zero, expect in zero were it is undefined, which makes it impossible to use as such with gradient techniques. Thus, other losses have been introduced as surrogates [Bis+95], like the mean squared error.

Gradient descent

As described in the previous section, learning the parameters of a model consists in finding those minimizing the loss function that quantifies the performance of the model. We denote here $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ a generic loss function that we assume continuously derivable and $\theta \in \mathbb{R}^n$ the parameter vector, argument of $\mathcal{L}$. The minimization of $\mathcal{L}$ consists in finding θ^* that minimize $\mathcal{L}$, if it exists:

$$\theta^* = \operatorname*{argmin}_{\theta \in \mathbb{R}^n} \mathcal{L}(\theta) \tag{2.5}$$

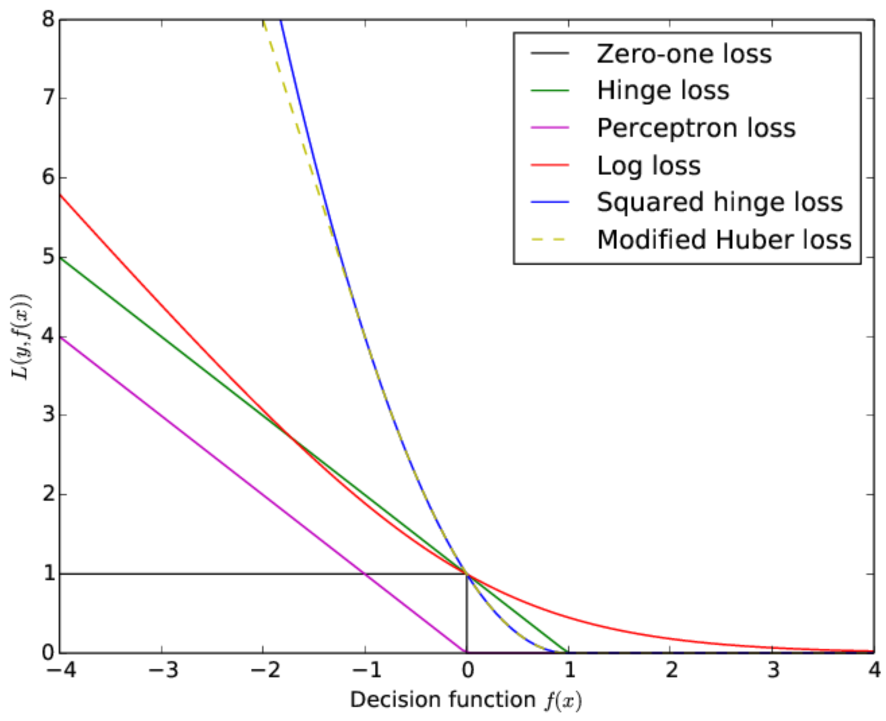


Fig. 2.2.: Representation of multiple loss functions in the context of binary classification. The x-axis is the output of the model and the y-axis is the value of the loss.

The gradient $\nabla\mathcal{L}(\theta)$ of $\mathcal{L}$ at point θ indicates the direction of steepest ascent around θ. The gradient descent algorithm presented in algorithm 2 exploits this property by taking small steps in the opposite direction of the gradient to iteratively lower the value of $\mathcal{L}$. The size of the steps is controlled by parameter γ, also called learning rate.

```
**Data**: $\theta_0, \gamma, p$
**Result**: $\hat{\theta}$
**for** $t \in \{1, 2, \dots, n_{epochs}\}$ **do**
    Compute $\nabla\mathcal{L}(\theta_{t-1})$;
    Takes a step: $\theta_t = \theta_t - \gamma\nabla\mathcal{L}(\theta_{t-1})$;
**end**
```

Algorithm 2: The gradient descent iteratively updates parameter θ along the direction of $\nabla\mathcal{L}$ to minimize the loss $\mathcal{L}$. The gradient $\nabla\mathcal{L}$ is computed using all training examples. Here, n_{epochs} iterations are made.

When using gradient descent, finding a global minimum is certain only if $\mathcal{L}$ is convex. In other cases, the only guarantee is that it will fall in a local minimum. The speed of the convergence depends on two parameters: the learning rate and the slope of the loss. Very often, the closer θ is to the minimum – whether local or global –, the closer the gradient $\nabla\mathcal{L}(\theta)$ is to 0. In practice, this means that the actual minimum is never reached. For instance on figure 2.3, the minimization of a simple non-convex loss is depicted. Two gradient descent runs with different initial parameters θ_0 lead to different solutions. Moreover, it illustrates well the slowdown due to the slope decrease around extrema.

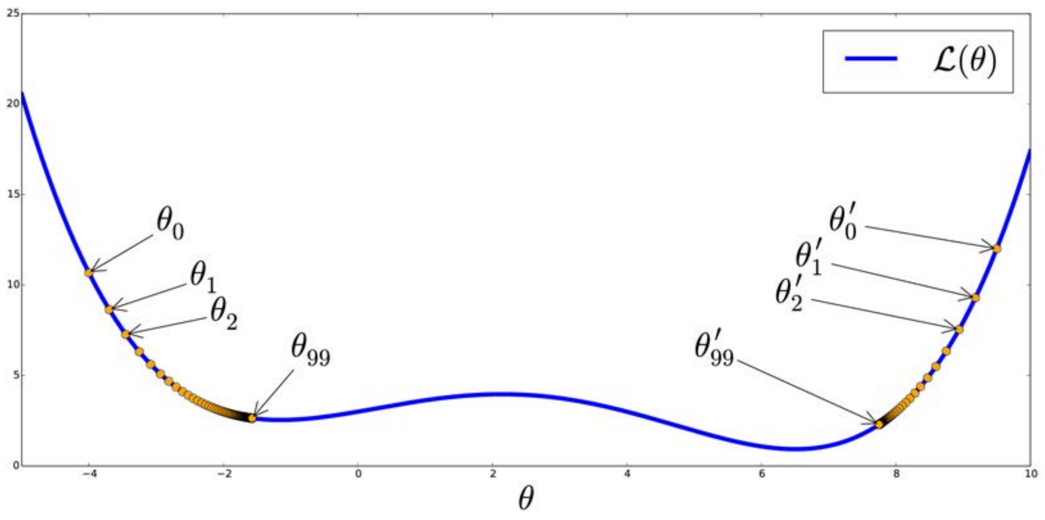


Fig. 2.3.: Example of a non-convex loss function and two runs of 99 iterations of gradient descent with different initializations.

Stochastic gradient descent

In machine learning, it is often the case that a loss $\mathcal{L}$ can be written as a sum of m functions $\mathcal{L}_i$:

$$\mathcal{L}(\theta) = \sum_{i=1}^{m} \mathcal{L}_i(\theta) \tag{2.6}$$

This is the case when the loss is the sum or average of a quality measure on each example. For the MSE in equation (2.4), we have $\mathcal{L}_i(\theta) = (y_i - g_\theta(\mathbf{x}_i))^2$. Gradient descents can be adapted to this special case by randomly selecting at each iteration one function $\mathcal{L}_i$ and only considering $\mathcal{L}_i$ for the update. For the steepest descent, that is taking a step in to opposite direction of $\nabla \mathcal{L}_i(\theta)$ only as depicted in algorithm 3.

For the MSE, the stochastic approach updates the parameters θ with respect to the squared prediction error for one random example of the training set at a time. As only one example is needed for each update, this is efficient for large-scale learning [LCB04]. The convergence of stochastic gradient approaches highly depends on using a decreasing learning rate [Bac14]. An intermediate approach is provided by mini-batch gradient descents, that computes stochastic updates on small batches of examples. A complete discussion on stochastic, mini-batch and batch gradient descents can be found in [Mon+12].

```
Data: $\theta_0, \gamma, p$
Result: $\theta_p$
for $epoch \in \{1, 2, \ldots, n_{epochs}\}$ do
    Randomly shuffle the training set;
    for $i \in \{1, 2, \ldots, m\}$ do
        Draw the next example;
        Compute $\nabla\mathcal{L}_i(\theta_{t-1})$ w.r.t to this example;
        Takes a step: $\theta_t = \theta_t - \gamma\nabla\mathcal{L}_i(\theta_{t-1})$;
    end
end
```

Algorithm 3: Stochastically updating parameter θ using n_{epochs} epochs to minimize the loss $\mathcal{L}$. For each epoch, all m training examples are seen in a random order.

2.1.3 Multi-Layer Perceptron

Neurons are the basic brick of neural networks that can be assembled in many different architectures so they can tackle different tasks. We will now study one possible way to do so: building layers and multi-layer perceptrons. The links between neurons imply links between parameters of each neurons and these links have to be taken into account when computing the gradient. This is commonly done using back-propagation.

Neuron layers

We define a layer as a stack of u units that will all have the same input $\mathbf{x}$ and activation function f. Each unit has its own set of parameters $\mathbf{w}$ and b. We stack all these parameters into a weight matrix $\mathbf{W} \in \mathbb{R}^{u \times n}$ and an intercept vector $\mathbf{b} \in \mathbb{R}^u$. As for neurons, we simplify our notations by using $\Theta = (\mathbf{W}, \mathbf{b})$. We denote $\mathbf{h}$ the activation of a layer – another name for its output – that is computed as:

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{2.7}$$

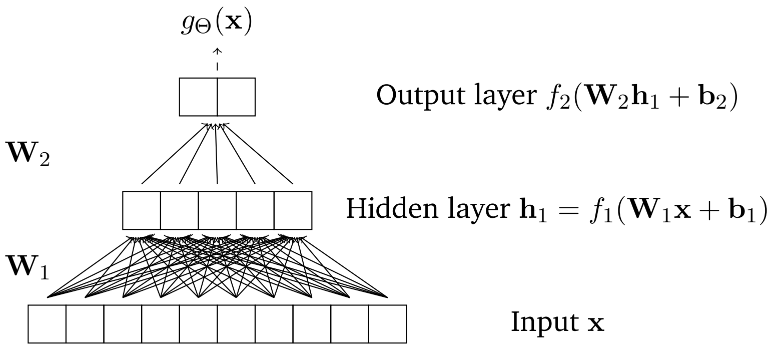
This time f is applied element-wise on the vector $\mathbf{W}\mathbf{x} + \mathbf{b}$.

A multi-layer perceptron (MLP) is a neural network composed of a succession of layers so that the input of a layer is the output of the preceding one. Subscript l indicates the parameters of the l-th layer of the network, starting at $l = 1$. Now Θ is used for all the parameters of the network and Θ_l for those of the l-th layer. Feeding the input forward in the network can be defined recursively:

$$\mathbf{h}_0 = \mathbf{x} \tag{2.8}$$

$$\mathbf{h}_l = f_l(\mathbf{W}_l\mathbf{h}_{l-1} + \mathbf{b}_l) \tag{2.9}$$

Figure 2.4 represents one such network. One can easily see by looking at this representation that the number of parameters depends on the input size and the number of units. In figure 2.4, they are less hidden units than input dimensions ($u < n$) but it is of course possible to have more ($u \geq n$). We denote by Θ all parameters of the network. The activation of the last layer is the output $g_\Theta(\mathbf{x})$ of the network. If the task is binary classification, only one real output is needed, thus the output layer should have only one unit. In the case of multi-class classification, a common approach is to use one output per possible class and a one-vs-rest approach [LC+90].



$g_\Theta(\mathbf{x})$
Output layer $f_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$
$\mathbf{W}_2$
Hidden layer $\mathbf{h}_1 = f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
$\mathbf{W}_1$
Input $\mathbf{x}$

Fig. 2.4.: Representation of multi-layer perceptron. From top to bottom the output, the activation of the output layer, the activation of the hidden layer and the input $\mathbf{x}$.

The benefit of this layer-wise architectures is that each layer learns a disentangled representation of the data as its activation. The representation of the last layer should be disentangled enough to enable an effective linear classification of the example, done by the output layer. The popularity and effectiveness of neural network is due to this ability to learn disentangling representations. However, these representations are hard to – humanly – interpret and can be considered as black boxes. Some intuition can be gained by studying inputs and outputs as in [Dea+12] but this should be considered carefully [Sze+13].

Back-propagation algorithm

Training multi-layer perceptrons is done using gradient descent. The computation of the gradient is straightforward when only one layer is used. However, it gets harder when more than one layer is used. The problem of minimizing a MSE (2.4) loss using a neural network with L layers is written in (2.10). Here, we denote θ_l the parameters $\mathbf{W}_l$ and $\mathbf{b}_l$ of the layer l and Θ the complete set of parameters (all layers).

$$\begin{aligned}\Theta^* &= \underset{(\theta_l)_l}{\text{argmin}} \frac{1}{m} \sum (y_i - g_\Theta(\mathbf{x}_i))^2 \qquad (2.10)\\ &= \underset{(\theta_l)_l}{\text{argmin}} \frac{1}{m} \sum (y_i - f_L(\mathbf{W}_L f_{L-1}(\dots f_1(\mathbf{W}_1 \mathbf{x}_i + \mathbf{b}_1)\dots) + \mathbf{b}_L))^2\end{aligned}$$

The back-propagation algorithm [Rum+88] computes the gradient of a loss function with respect to the parameters of all layers. Mathematically speaking, it uses the chain rule of derivatives. It computes the gradient $\nabla_L \mathcal{L}$ of the loss with respect to the parameters θ_L of the last layer. To obtain the derivative for the previous layer, it computes the derivative $\nabla_{L-1}\theta_L$ of the parameters θ_L of with respect to the one of the previous layer θ_{L-1} and multiplies the two quantities: $\nabla_{L-1}\mathcal{L} = \nabla_L \mathcal{L} * \nabla_{L-1}\theta_L$. The rule is iterated up to the parameters of the first layer of the MLP. This algorithm is called back-propagation as it acts by propagating the error backward in the network. The algorithm works independently of the output dimension.

Gradient vanishing

Training a neural network in not easy. Many parameters are involved in the algorithm. The learning rate γ that we introduced in algorithms 2 and 3 was fixed. It is possible to use decreasing learning rates or adaptive ones. Extensions of the gradient descent such as the Wolfe method (first order) or Newton methods (second order) provide (computationally expensive) ways of finding good learning rate at each iteration. We presented algorithms using a fixed number of iterations when many stopping criteria are possible. It is possible to stop when the delta in loss function after an update is small enough or to use the loss on a validation set. Finally, choosing between batch, mini-batches and stochastic is not straightforward. These problems are highly dependent on the usage context of the network and we will not discuss them here. More information on those subjects can be found in the complete review provided by [Mon+12].

Another difficulty arises from the concept of back-propagation: gradient vanishing. Through the update rules of back-propagation, the output layer absorbs a great part of the prediction errors. Only a small part is reported on the previous layer that, in turns, absorbs a great part of it. This makes MLP with more than two layers hard to train since most parameters in the lower layers will hardly have changed from their initial values. One solution has been found by using unsupervised learning and auto-encoders. Auto-encoders are a kind of MLP, composed of only one hidden layer and one output layer, trained in a unsupervised manner, as described in the next section. A first solution for training deep neural network architectures composed of multiple layers has been to stack several auto-encoders as explained in section 2.2.1. Note that recent progress in the field involving weighted initialization procedures and alternative neuron units now allow the training of large neural network architecture from scratch.

2.2 Unsupervised learning and neural networks

Neural networks have not only been used in the context of supervised classification and can be trained in an unsupervised way. For instance, Kohonen maps [Koh82] are a type of artificial neural network, trained to learn low-dimensional representations of inputs. We will focus here on auto-encoders that are known to be effective to extract text representations. We will begin by presenting them as well as their unsupervised training procedure. Then we will present how to use them to overcome the gradient vanishing problem and train deep neural networks.

2.2.1 Auto-encoders

The multi-layer perceptrons presented in section 2.1 where trained to correctly predict the label of each example. Auto-encoders are trained to predict the example itself. The idea is to consider auto-encoders as a coding function – input to latent spaces – and a decoding function – latent to input spaces. They are representation learning techniques. In fact, linear auto-encoders – using linear activation functions – are tightly linked to Principal Component Analysis [RM98]: they learn a similar representation space.

Reconstruction loss

In this thesis we will focus on the deterministic framework using back-propagation to train auto-encoders. However, other frameworks have been proposed like the probabilistic Restricted Boltzmann Machines [HS06] trained using contrastive divergence [CPH05]. An auto-encoder is represented on figure 2.5, in which a fully-connected layer is only represented by 2 arrows.

We have seen in section 2.1.2 that the quality of a prediction is quantified by a loss function and that learning the parameters of a neural network consists in minimizing this loss function. For auto-encoders, the prediction is called reconstruction of input and should be close to the original input. For real output variables, this *closeness* is generally quantified by the MSE (2.4). Other loss functions can be used such

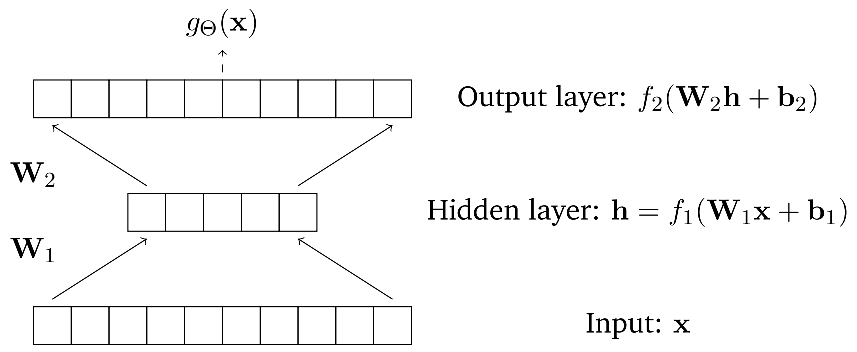

$g_\Theta(\mathbf{x})$
Output layer: $f_2(\mathbf{W}_2\mathbf{h}+\mathbf{b}_2)$
$\mathbf{W}_2$
Hidden layer: $\mathbf{h} = f_1(\mathbf{W}_1\mathbf{x}+\mathbf{b}_1)$
$\mathbf{W}_1$
Input: $\mathbf{x}$

Fig. 2.5.: Representation of an auto-encoder. From top to bottom the output, the activation of the output layer, the activation of the hidden layer and the input. Auto-encoder are trained to reproduce as their output the input they are presented.

as the cross-entropy for binary data. An auto-encoder is composed of two layers: an hidden and an output layer and, under MSE, the reconstruction loss is:

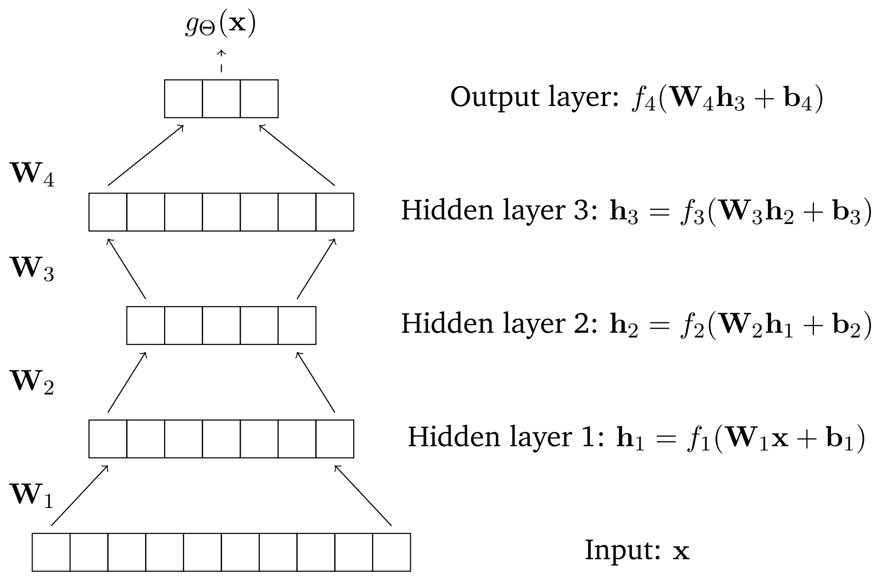
$$\begin{aligned}\Theta^* &= \operatorname*{argmin}_{(\theta_l)_l} \frac{1}{m}\sum(\mathbf{x}_i - g_\Theta(\mathbf{x}_i))^2 \\ &= \operatorname*{argmin}_{(\theta_l)_l} \frac{1}{m}\sum(\mathbf{x}_i - f_2(\mathbf{W}_2 f_1(\mathbf{W}_1\mathbf{x}_i + \mathbf{b}_1) + \mathbf{b}_L))^2\end{aligned} \tag{2.11}$$

As said in the last section, the back-propagation algorithm works independently of the output size. Training an auto-encoders simply means replacing the target $\mathbf{y}_i$ by the input $\mathbf{x}_i$ in the derivations. The back-propagation algorithm remains the same.

Stacked auto-encoder

Auto-encoders are a well known kind of neural network, mainly used in the past to learn compressed representation of the data points. They can also easily be stacked, in the same way as layers are stacked in multi-layer perceptrons. The first auto-encoder is trained to reconstruct the input data. It gives a first representation of the data with its hidden layer. The second auto-encoder is then trained to reconstruct the activation of the hidden layer of the first auto-encoder. This process is iterated until all auto-encoders are learned. Figure 2.6 represents a stacked auto-encoder. The first layer is trained to reconstruct the input $\mathbf{x}$, the second layer is trained to reconstruct $\mathbf{h}_1$, and so on.

Stacked auto-encoders form deep neural networks: hidden layers are connected to each other, removing the output layer. This structure can then be fine-tuned for classification tasks as in [HS06] or [Glo+11]. Using auto-encoders in such a way can overcome the problem of vanishing gradients presented in 2.1.3. In



$g_\Theta(\mathbf{x})$
Output layer: $f_4(\mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4)$
$\mathbf{W}_4$
Hidden layer 3: $\mathbf{h}_3 = f_3(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3)$
$\mathbf{W}_3$
Hidden layer 2: $\mathbf{h}_2 = f_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2)$
$\mathbf{W}_2$
Hidden layer 1: $\mathbf{h}_1 = f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$
$\mathbf{W}_1$
Input: $\mathbf{x}$

Fig. 2.6.: Representation of a deep neural network for classification built stacking auto-encoders. From top to bottom the output, the activation of the output layer, the activation of the hidden layer and the input. Auto-encoder are trained to reproduce as their output the input they are presented. The last layer is an output layer, similar to these of traditional MLPs.

[HS06], each auto-encoder (trained using the probabilistic framework of RBMs) compresses the dimension of the representation space on images. In [Glo+11], the same principle is applied for texts. In both cases, the network is fine tuned at the end by adding a final classification layer and using back-propagation. Auto-encoders are considered to provide a good weights initialization for the final fine-tuning.

Learning representations

As presented above, hidden layers of auto-encoders learn coding functions mapping input data into a latent representation space while the output layer learns the decoding operation. When using non-linear activation functions, such as the logistic function (equation (2.12)), the coding and decoding operation are harder to interpret.

$$s(x) = \frac{1}{1+e^{-x}} \tag{2.12}$$

To understand the behavior of the network, it is possible to determine the input vector $\mathbf{x}$ that maximizes the activation of each neuron. This method was used in [Le+13] in which a deep neural network is trained using an impressive amount of computing power on images extracted from YouTube. The authors argue that neurons of deep layers learn high-level features able to detect generic concepts such as cat faces, human bodies or human faces. However, the recent study [Sze+13] reveals that this analysis methods may lead to false conclusions and argue that deep neural networks are black box models.

The ability of deep neural network to outperform state-of-the-art methods for tasks such as object, speech and hand-writing recognition comes from their high expressiveness. This expressiveness often leads to over-fitting. It has to be dealt with either by using a complexity control mechanism or by introducing prior knowledge about the task or the data or by a combination of the two. This is often addressed using some form of regularization and this is the setup adopted in this thesis.

2.2.2 Regularization

Consider a dataset lying in a n-dimensional input space and an auto-encoder with a hidden layer composed of k hidden units. This models has a great number of parameters: $\mathbf{W}_1 \in \mathbb{R}^{k\times n}$, $\mathbf{b}_1 \in \mathbb{R}^k$, $\mathbf{W}_2 \in \mathbb{R}^{n\times k}$, $\mathbf{b}_2 \in \mathbb{R}^n$. It amounts to $2kn + k + n$ parameters and the training may lead to over-fitting. In the context of classification, over-fitting means that the frontiers the model sets between classes are more representative of the distribution of the training data points than of the actual classes distribution. Similarly, for representation learning, it means that learned representations will be too specific to training data and not generic and robust enough for good generalization.

A first form of regularization used in this work is that we choose is to limit ourselves to simple auto-encoders (one hidden layer). Many models using only one layer (like auto-encoders do) are compared to deeper architectures on multiple datasets in [Coa+11] and one-layered models are as good as those deeper models for classification tasks. A second form of regularization we will us is to impose $\mathbf{W}_1 = \mathbf{W}_2^t$ where $\mathbf{W}^t$ is the transpose of $\mathbf{W}$. This framework is often used in the literature and is called tied-weights as the weights of both layers are tied and limits the number of parameters to train. Both forms are using the same regularization mechanism: they limit the number of parameters in the model.

We will now present regularization using $L2$ and $L1$ norms of the parameters. Then, we will look at constraints forcing the auto-encoder to learn sparse representations.

$L2$ and $L1$ norms

For neural networks, regularization using $L2$ and $L1$ norms is done by adding a term to the loss $\mathcal{L}$, the p-norm of the weights matrix $\mathbf{W}$, to the power p:

$$\Theta^* = \underset{\Theta}{\text{argmin}} \; \frac{1}{m} \sum_{i=1}^{m} \|\mathbf{x} - f_2(\mathbf{W}_1^t f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)\|_2^2 + \lambda \|\mathbf{W}_1\|_p^p \tag{2.13}$$

Such a term mechanically limits the possible values for the parameter $\mathbf{W}$. The intercept takes into account how badly centered the data is and is never regularized nor tied. $\lambda \in \mathbb{R}_+$ controls the importance of the regularization term. It profoundly impacts the quality of the result and has been extensively studied. We refer to [Mon+12] for more details on the meaning of regularization.

$L2$ regularization, also known as Tikhonov regularization in other domains [TA77], uses the $L2$ norm of the parameter. It is the Euclidean norm for vectors and the Frobenius norm for matrices denoted $\|\|_F$ in this case and defined as:

$$\|\mathbf{W}\|_F = \sqrt{\sum_{i,j} W_{ij}^2} \tag{2.14}$$

As presented in (2.13), the squared norm is used to avoid the derivation of the squared root. $L2$ can been either interpreted under the framework of Tikhonov's work on ill posed problems or as a Gaussian prior on the parameter distribution. It forces values in $\mathbf{W}$ to have a smooth distribution: a high variance in values of $\mathbf{W}$ is a symptom of over-fitting. In practice, it restricts possible values of $\mathbf{W}$ and, mechanically, the decision functions that the model can learn.

$L1$ regularization uses the $L1$ norm:

$$\|\mathbf{W}\|_F = \sum_{i,j} |W_{ij}| \tag{2.15}$$

$L1$ regularization has been popularized by the Lasso algorithm [Tib96]. It aims at reducing the number of input dimensions to fit by selecting the relevant ones. Weights of irrelevant dimensions are set to 0, this leads $\mathbf{W}$ to have only a few non-zero entries: $\mathbf{W}$ is then a sparse matrix. This is a form of feature selection. For feature selection, the $L1$ norm is a better choice than the $L2$ norm as shown in [Ng04] However, the $L1$ norm is not differentiable. This is a drawback as sub-gradients have to be used for the optimization making $L1$ regularization harder to use in practice.

It is possible to use both norms at the same time. This configuration is called the Elastic Net [ZH05]. It is used in [Raf+12] in the context of sentiment classification.

The high dimensionality of bag-of-word representations provides an adequate context for feature selection using the $L1$ norm. At the same time, the weight variance due to the high variability of term frequencies can be controlled using the $L2$ norm. As a result, in [Raf+12], sentiment classifiers regularized with the ElasticNet are better than those which are not.

Sparse coding

In the field of signal processing, $L1$ regularization has also been used for learning sparse representations, as, for instance, in compressed sensing [Don06]. Sparseness is also frequently used in dictionary learning [Mai+10], matrix factorization [Hoy02; Hoy04] and neural networks [Lee+06; Ben+09]. It has also been developed for probabilistic models such as RBMs [Lee+08]. Sparse coding concerns all the works studying the learning of sparse representation. Sparseness is a way to compress signals. Compressing signals means understanding them enough the focus on relevant parts. This is widely exploited in the field of signal processing, for instance with the minimum description length principle [Bar+98].

We denote $\mathbf{h}$ the learned representation of an input vector $\mathbf{x}$. In the context of an auto-encoder, it corresponds to the activation of the hidden layer. As presented in details in [Lee+06], a sparse representation is such that each vector $\mathbf{h}$ has only a few non-zero entries. One of the many possible approaches to learn sparse representations with neural networks is to use $L1$ regularization on $\mathbf{h}$ values:

$$\mathcal{L}(\Theta) = \frac{1}{m}\sum_{i=1}^{m} \|\mathbf{x}_i - g_\Theta(\mathbf{x}_i)\|_2^2 + \lambda\|\mathbf{h}_i\|_1 \tag{2.16}$$

This additional terms modifies the computing of the gradient but back-propagation straightforwardly adapts to it.

Sparseness can also come naturally from the data. For instance, in chapter 6, the learned representation $\mathbf{h}$ represents the distribution of activities in a user profiles. These activities explain the subway trips of the user. Among all activities shared by the 600k users present in our application, the individual profile of each user can only contain a few. Thus a sparse representation – sparse values of $\mathbf{h}$ – is a natural choice here.

2.2.3 Auto-encoders for classification

The representation learned by auto-encoders are used for subsequent tasks such as clustering, regression or classification. This section focuses on presenting an adaptation of these neural networks to the latter. We will begin by applications of auto-encoder for supervised classification. We will then present how to benefit from supervision while learning the parameters of the auto-encoder. Finally, it is often the case that not all training examples are labeled and we will present how to adapt auto-encoders to semi-supervised tasks.

Supervised classification

As presented in section 2.1, supervised classification includes:

Binary classification There are 2 possible classes. Examples are either in one or the other.

Multi-class classification There are more than 2 possible classes. Examples belong to only one class.

Multi-label classification Examples can belong to multiple classes.

The idea behind representation learning is to exploit the regularities in the dataset to build robust features. The long term goal behind this is to replace the tedious design of hand crafted features with learning features. [Coa+11] compares the quality of representations learned by different algorithms for image classification. Object recognition is also a classification task where the goal is to predict which object appears on an image. Unsupervised training of neural networks leads to the extraction of robust representations that can then be exploited for object recognition [Ran+07]. Another application to image classification tasks is presented in [Rif+11b; Rif+11a]. Authors exploit the plasticity of gradient based learning to add terms that forces the representation to take into account the manifold supporting the data points.

Supervised auto-encoder

If the goal is supervised classification and the dataset is fully supervised, then it is possible to consider a supervision loss in addition to the reconstruction loss of an auto-encoder. It can be formulated as adding a classifier on top of the hidden layer, next to the output layer of the auto-encoder as presented in (2.17). We denote f_s and $\Theta_s = (\mathbf{W}_s, \mathbf{b}_s)$ the parameters of this classifier. The parameter λ_s is used to balance the reconstruction and classification in the loss:

$$\mathcal{L}(\Theta) = \frac{1}{m} \sum_{(\mathbf{x},\mathbf{y})} \|\mathbf{x} - f'(\mathbf{W}^t\mathbf{h} + \mathbf{b})\|_2^2 + \lambda_s \|\mathbf{y} - f_s(\mathbf{W}_s\mathbf{h} + \mathbf{b}_s)\|_2^2 \tag{2.17}$$

In [BL+12], a similar term is successfully used in the context of musical pitch identification to improve the quality of the learned representations for the task at hand. It has also been used in [RL13] to correctly recognize handwritten digits as an additional constraint on a recurrent auto-encoder. Supervised representation learning models in [BL+12; RL13] are found significantly better than classifiers operating on raw inputs and classifiers trained on representation learned in a purely unsupervised manner.

Semi-supervised auto-encoder

Supervised data is however expensive. User reviews represent an exception where huge quantities of fully supervised data points are available. In the context of object recognition or scene labeling, images have to be hand labeled, which can be expensive when downloading huge quantities of unlabeled images from the Internet is not. This leads to datasets where only some data points are labeled, the others are unlabeled. Models exploiting both labeled and unlabeled examples are semi-supervised.

Semi-supervised auto-encoder have been extensively studied in the literature. In [RS08], stacked auto-encoders are trained using a supervision criterion similar to (2.17) for document classification. When the number of available labeled examples is low, they found that semi-supervised auto-encoders outperform semi-supervised SVM trained either on raw data or on the output of unsupervised auto-encoders.

2.3 Different deterministic architectures

The previous sections, 2.1 and 2.2, have provided background material on neural networks, how to train them and how to use them to learn representations. They have been deliberately focused on multi-layer perceptrons and auto-encoders as these are the main techniques used in the thesis. However, other network architectures exist. We describe below some of these extensions to the basic MLPs. We will begin this review of recent advances in neural networks by presenting recurrent neural networks. We will then present auto-encoders in the context of supervised classification and we will finish this review by presenting specific architectures used to learn text representations.

2.3.1 Recurrent neural networks

In vanilla MLPs, neurons are organized in layers and each layer is only connected to the one before and the one after it. There is no connection inside the layer nor towards other layers. This means that the output of a network is only depending on the current input and not on previous ones. This is a strong limitation in the context of sequential datasets and more generally for time dependent analysis. Recurrent architectures – networks with backward connections – have been proposed for these problems. Recurrent Neural Networks (RNN) are not used in the main topic of this thesis so we only briefly present them here. Our goal is to show the plasticity of the learning algorithms used in this thesis and to present an example of time-sensitive models. The reference [Gra12] provides a review of RNN.

Principle

As presented above, the main difference between MLPs and RNNs is that connections between neurons of RNNs can form loops. We will present the principles and implications of RNNs by limiting ourselves to simple recurrent MLPs. We consider a network with one hidden layer that is self-connected and refer to [Gra12] for more material. This self-connection means that the activation of this layer for one input also depends on its activation for the previous input.

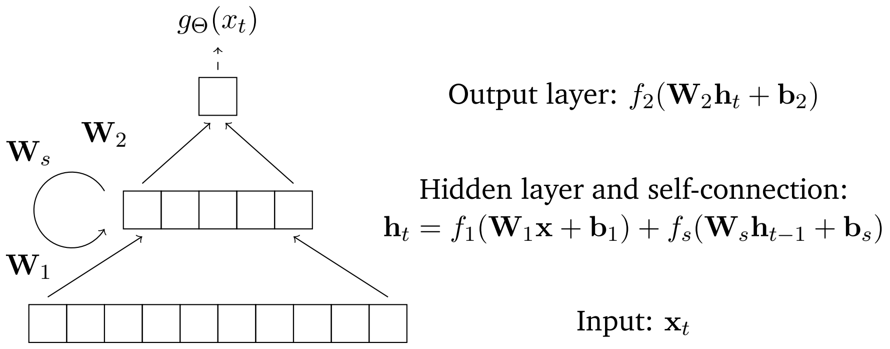

$g_\Theta(x_t)$
Output layer: $f_2(\mathbf{W}_2\mathbf{h}_t + \mathbf{b}_2)$
$\mathbf{W}_2$
$\mathbf{W}_s$
Hidden layer and self-connection:
$\mathbf{h}_t = f_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + f_s(\mathbf{W}_s\mathbf{h}_{t-1} + \mathbf{b}_s)$
$\mathbf{W}_1$
Input: $\mathbf{x}_t$

Fig. 2.7.: Representation of a recurrent neural network. From top to bottom the output, the activation of the output layer, the activation of the self-connected hidden layer and the input.

Formally, this introduces a new set of parameters denoted $\theta_s = (\mathbf{W}_s, \mathbf{b}_s)$. The activation function of this connection is denoted f_s. We also introduce *time* t to distinguish the current input from the previous $t-1$ one. The activation of the hidden layer is denoted $\mathbf{h}$ and is computed as:

$$\mathbf{h}_t = f(\mathbf{W}\mathbf{x}_t + \mathbf{b}) + f_s(\mathbf{W}_s\mathbf{h}_{t-1} + \mathbf{b}_s) \tag{2.18}$$

Such a connection introduces memory in the network since predictions depend on what was seen in the past. The back-propagation needs to be adapted to this new situation. Back-propagation through time (BPTT) [Wer90] provides an adaptation of the back-propagation algorithm to compute the gradient of parameters for a complete sequence of examples $(\mathbf{x}_t)_t$. The learning principle is however the same as for MLPs: a gradient descent. This shows the plasticity of this algorithm that can be adapted to many different settings. It also means that RNNs benefit from this plasticity and can be augmented as classic non recurrent networks with other constraints. Note that in figure 2.7, we have considered recurrent connections with a delay of 1 – prediction at t depends on $t-1$. The same BPTT can be extended to other delays – $t-1, t-2, \ldots$.

Applications

RNNs have been used for different tasks for processing sequential data. On of these tasks is sequence labeling, the main topic of [Gra12]. It consists in attributing a discrete set of labels to a sequence of observations. These labels are often associated to states of the sequence. In the context of handwriting recognition, the observations can be the position of the ink on the paper and the labels are the letters and the states are the recognized letters or words. RNNs proposed in [Gra12] are considered as state-of-the-art for sequence labeling. The recent [PC14] successfully applies RNNs to scene labeling. The study of time-series has long been another domain of interest for RNNs [Dor96].

The data used in this thesis are coming from timed observations. However, we use neural network to extract stable generic representations of the topics discussed in user reviews. We make the assumption that the distribution of such topics is more dependent of the item reviewed or of the reviewer than on the time of the review. Thus, we chose not to use models such as recurrent neural networks that could take previous reviews directly into account and to use the ability of auto-encoder to benefit from regularities in the available reviews to learn robust representations. The work of [ML13b] shows that taking into account how experienced the reviewer is concerning the domain of the item to review – wine or beer tasting or music for example – is beneficial for recommender systems. Considering time could provide interesting extensions of our work.

2.3.2 Architectures for text representations

Representation learning has become a field of intensive research in NLP and text mining. The former provides a quantity of hard tasks such as named entity recognition or part of speech tagging. Recent works like [Col+11] exploits the co-occurrences of words in large text corpora to infer semantical and syntactical properties on words and obtain performance equivalent to state-of-the-art NLP models without using prior knowledge as the latter do. For the latter, text representations were most of the time based on bag-of-words or n-grams representations[1]. These raw representations can be compressed by grouping words together in topics. These topics can be obtained by training auto-encoders on bag-of-words [Glo+11]. Other approaches have also been proposed for learning topic representations like the probabilistic approach of Latent Dirichlet Allocation [Ble+03].

Working units: texts, sentences or words

One of the main difficulty with text is to find the correct working unit. User reviews, for instance, are short texts and [Glo+11] considers them as a whole assuming that the overall polarity is more important than the polarity of each sentence. However, in [Gan+09], authors take the opposite approach focusing on the polarity of each sentence. The work on [Soc+11] proposes words as the working unit by learning a latent representation for each word. This is the finest granularity that can be considered for representation learning –letters would make little sense. It assumes that is is possible to learn a representation space where semantically close words are close. However this leads to a problem when representing sentences: sentences can vary in number of words. Traditional machine learning models assume that the input space has a fixed size. To overcome this issue [Soc+11] trains recursive auto-encoders using a tree like structure that respects the word ordering of each sentence and produce a fixed length representation for it. In [LM14] a latent representation of sentences or paragraphs is learned additionally to the representation of words.

In this thesis, we will use auto-encoders to extract representation for text reviews in chapter 4. As we said, they are typically short texts and express the overall opinion of users on items. We want to select sentences relevant to each users. In this context, working with topics gives more abstraction than working directly on words. This is why we chose an approach similar to [Glo+11]. However, the work of [LM14] provides insights for possible extensions of our work.

Multi-task representations

The impressive work in [Col+11] proposes to tackle multiple difficult Natural Language Processing (NLP) tasks using one unified neural network. All tasks are labeling tasks that are traditionally performed using hand-crafted features and SVMs. [Col+11] proposes to replace the tedious design of these features. The main idea is to store vector representations for words in look-up tables.

[1]A review of these representation and preprocessing is done in 5.1.2

Figure 2.8 represents the first architecture introduced in [Col+11]. The input layer is the initial representation of words: each word is represented by a set of indexes. One index is the index of the word itself in the vocabulary but other are possible: index of the lowered-case stem of the word in the corresponding dictionary, index for the capitalization of the word among all its possible capitalizations, ... Each word is described as multiple features and each feature is represented as a vector, stored in a look-up table. To obtain a vector representation of each word, the vectors of each feature of this word are concatenated. This is done, in this first architecture for all words of a window of fixed length around the word. The vectors representing all words of the window are then concatenated in one large vector. This vector serves as the input to a MLP composed of one hidden layer, with hard hyperbolic tangent[2] as its activation function, and one linear output that predicts the label for the word of interest.

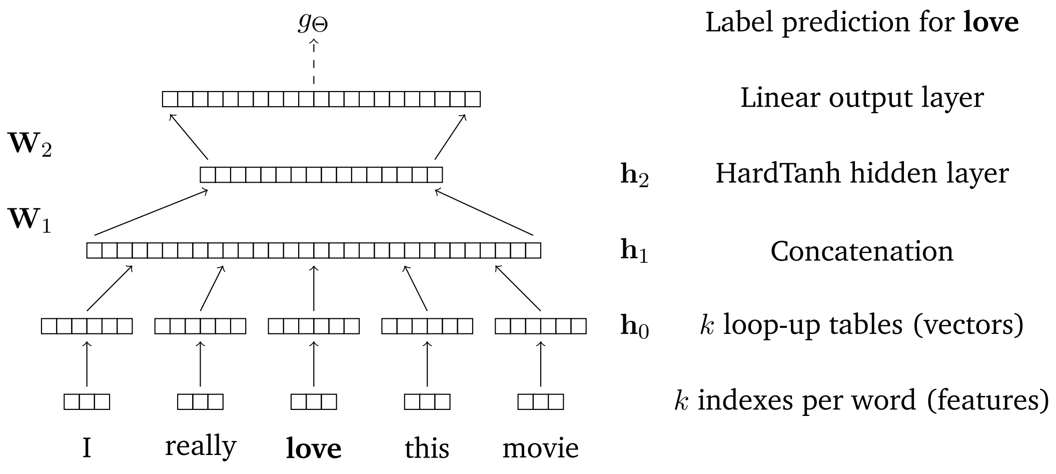


Fig. 2.8.: First architecture of [Col+11]: predicting labels for the word of interest, **love**, using a fixed size context window. Words are described by the concatenation of k features, each feature is a vector stored in a look-up table. A neural network is trained on the concatenation and is composed of a HardTanh hidden layer and a linear output. The prediction g_Θ is the set of labels for the word of interest, **love**.

Now, the use of a window around the word of interest, love in figure 2.8, was an heuristic to avoid considering the complete sentence that contains the word. Considering this sentence is more complicated as sentences have variable lengths. To overcome this, [Col+11] proposes to replace the concatenation step by a convolution and candidate wise a max over time. The convolution gives a set of vectors of the same length and the max over time takes the maximum for each dimension of these vectors. The result is a vector that always has the same length, the output dimension of the convolution.

Training these networks is done by back-propagations and [Col+11] pre-trains the look-up tables in an unsupervised way before. They obtain performances that com-

[2]The hard hyperbolic tangent is a linear approximation of the hyperbolic tangent that provides faster computations.

pete with state-of-the-art methods. The benefit of this work was to show that many different tasks in NLP could be achieved using machine learning and more specifically using large size neural networks. It also proved the potential of learning representations for text. However, because of their complicated training such architectures remain challenging to use in practice. Also, the model is a black box, understanding the decision is hardly possible.

Word2vec and paragraph vectors

Word2vec [Mik+13] is a neural network that aims at learning robust representations of word that embed semantic information. Words that are close in the representation space should be close in semantics (like *Berlin* and *Paris*). One possible architecture of word2vec is presented in figure 2.9. Here the goal is to predict the last word (here *movie*) of parts of sentences of fixed length (here *I really love this movie*), given the first words (here *I*, *really*, *love* and *this*). Representations $\mathbf{h}_0$ of words are stored in look-up tables. They are concatenated or averaged into one unique vector $\mathbf{h}_1$ that is used to predict $\mathbf{h}_2$ that should be close to the representation of the last word. Another architecture is to predict the window given the last word. As explained in [Mik+13], robust representations are learned by alternating the various architectures on large text datasets. This make training such models complicated even if the implementation is available.

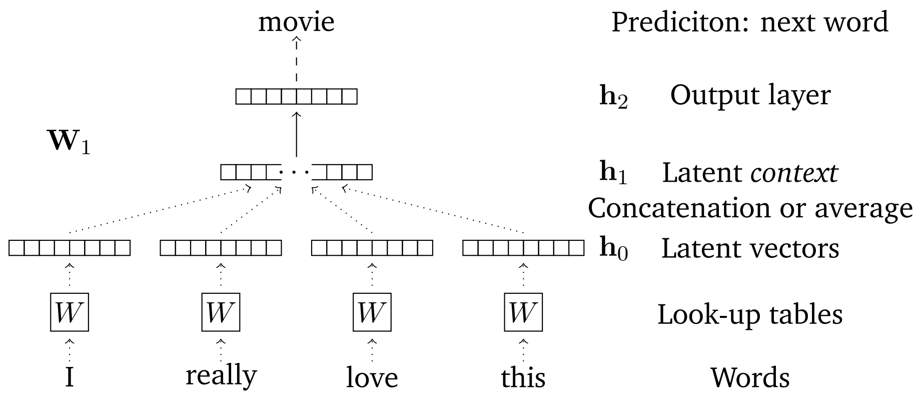


Fig. 2.9.: One possible architecture of word2vec: predicting one word based on those before it and the paragraph

Paragraph vectors [LM14] is an extension of the word2vec model [Mik+13]. The extension in [LM14] is to also learn a representation for paragraphs – also in a look-up table – as well as for words. Figure 2.10 represent a neural network aiming at predicting the latent representation of a word, *movie*, using the ones of the 3 words before it – *I*, *love*, *this* – and the one of the paragraph they are part of.

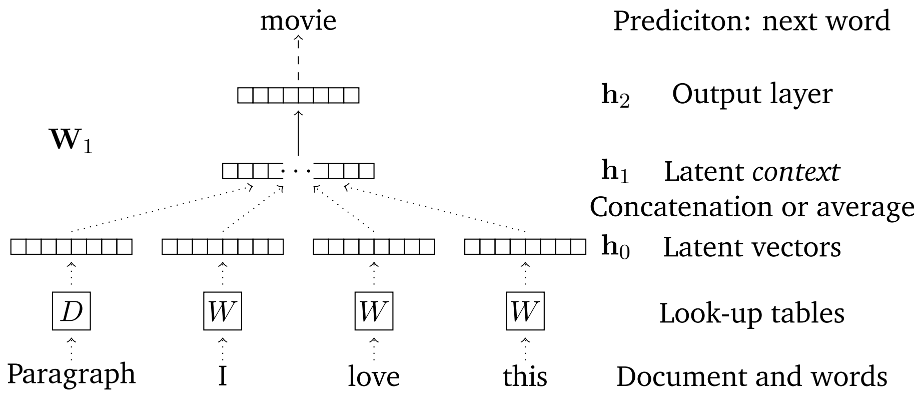


Fig. 2.10.: One possible architecture of paragraph vectors: predicting one word based on those before it and the paragraph

Different architectures are proposed in the two papers cited above: predicting the middle word of the window based on those around, predicting one word based on the paragraph only, predicting words before based on the last word of the window, . . . As explained in [LM14], to correctly train the model it is necessary to use large datasets and a combination of all architectures. Also, there is no procedure but heuristic experimentation to select parameters such as the size of the latent representation or the window. Because of this, training such models is tedious. Moreover, learned representations are very difficult to interpret.

2.4 Conclusion

In this chapter, we provided background material on neural networks. We started from the basic units that compose them, neurons [Ros58], and how to assemble them in layers and multi-layer perceptrons [LC+90] or auto-encoders [Vin+08] and how to train them. This is one framework – a deterministic one – for neural networks and other have been proposed, like the probabilistic RBMs [HS06]. We chose it because back-propagation provides a simple yet powerful, plastic and elegant algorithm to train these models. To control the complexity of neural networks, we chose to use regularization and presented $L1$ and $L2$ regularization. For more detailed information on how to train neural networks, we referred to [Mon+12].

In this thesis, we will apply auto-encoders to learning representations for texts in chapter 4, which has become a field of intensive research. We motivated our choice of learning with auto-encoder, like in [Glo+11], topics out of raw text representation to represent interest of users. We also presented recent advances [Col+11; LM14] in the domain that provide insights for future work.

3 Learning representations with matrix factorization

Contents

In data analysis and machine learning, two approaches can be distinguished. A first family of models tries to extract a dictionary of explanatory functions, the number of which is typically hand-fixed. This is the case of k-means, a popular clustering model [KR09], which computes k prototypes from the data and clusters data around them using similarities. These k prototypes form a dictionary. The principal component analysis (PCA) [Pea01; Hot33] computes k components from the data that, again, form a dictionary. A second family considers a fixed explanatory dictionary and, this time, looks for a code explaining, for each example, the contribution of these dictionary elements. This approach allows, for example, to compress a representation, as it is the case in compressed sensing where a fixed dictionary is used to decompose a signal. The latter is then represented by its code: the contribution of each dictionary element to the signal. If the dictionary is relevant, only a small number of basic elements contributes to the signal and its code will only have a few non-zero values, achieving compression. This is a common approach to compress images [Hor+12]. Matrix factorization methods propose to unify these two families by simultaneously learning a relevant dictionary and the associated code. They differ from methods learning only the dictionary suc as PCA or k-means for which the code is simply infered in a second step by projecting the data onto the dictionary basis. For the PCA, the data matrix is projected on the dictionary of k principal components without control over the quality of reconstruction. For k-means, the code is simplified in the allocation step search of the nearest centroid [1]. Unlike projection methods such as compressed sensing or the random projection [BM01], matrix factorization also learns the dictionary.

Matrix factorization techniques are named this way as they learn the product of explanatory factors, usually two factors (the dictionary and the code matrix), that best approximate a data matrix. They take root in the statistical analysis of data with the PCA [Pea01; Hot33] and in linear algebra with SVD [GVL12]. Matrix factorization allows to extract interpretable representations out of the data. This is particularly the case for the blind source separation problem for which matrix factorization provides state of the art models. In this setting, an observed signal must be explained as resulting the combination of hidden sources. For a musical recording, for instance, the observed signal can be broken down into several sources: the contributions of each instrument. The sources are the elements of the dictionary and the contribution of each source is the code. Learning both the dictionary and the code enables learning both the optimal representation space and the optimal encoding of data in this space. Interestingly, many problems can be seen as blind source separation problems and thus find a natural solution using matrix factorization techniques. aezThis is the case for facial recognition [PZ11]: each face, the observed signal, is considered to be composed of several sources, the possible face parts (different eyes, different noses . . .). Recommender systems can also be seen as blind source separation problems and use matrix factorization techniques through the collaborative filtering framework [KB11]. They work with dyadic data [Hof+99], data generated by links between two types of entities: users and items. The ratings are the observed signals that can be decomposed on a set of hidden sources, the user and item profiles.

For all these applications, matrix factorization propose to learn the dictionary and the code at the same time. However, this is a difficult learning problem. As the number of parameters involved in the factorization is large, matrix factorization techniques are highly sensitive to small perturbations in the data and have a high variance. Without complexity control mechanisms, e.g. regularization, learned solutions are doomed to be too specific to the observed data and generalize poorly. The complexity control mechanisms involved in matrix factorization limit the number of possible configurations, which increases the bias and limits the variance. A first approach is to limit the rank of the reconstruction matrix, the product of the explanatory factors, and is known as the search of the best low-order approximation of a matrix, the solution of which is given by the Singular Value Decomposition

[1]This corresponds to seek only the maximum amplitude in the code element

(SVD) [GVL12] To control the variance of matrix factorization, the challenge is to prohibit configurations that do not make sense with respect to the nature of data. Principal Component Analysis (PCA) [Pea01; Hot33] explains each example as a linear combination of basis vectors by imposing the constraint that these components must not be correlated. PCA is typically used for visualization purposes, by choosing a two or three dimensional representation space, and for noise removal [Tho+02], hypothesizing that noise is not correlated with the signal. Independent Component Analysis (ICA) [Com94] forces components to be independent, which makes it the reference method for the blind source separation problem [Ama+96] as sources often are assumed independent. Non-negative matrix factorization [LS01] imposes that each factor is composed only of non-negative values. It is a reference model for facial recognition as it successfully decomposes faces in parts [LS99]. All these constraints limit the number of different solutions of the factorization and mechanically increase the bias and limit the variance. Hence the need to find appropriate constraints for each problem, to limit the variance while preserving the *best* factorizations. It is also possible to use, in addition to or instead of these constraints, a regularization framework. For example, sparse matrix factorization [Hoy02; Hoy04] forces factors to have few non-zero values (sparse matrices) using $L1$ regularization. They take root in dictionary learning [Mai+08; Mai+10], a field of image processing where the images, highly dimensional data, are compressed using dictionaries of visual words. These dictionaries were initially hand fixed, but they are now learned from the data [Mai+08; Mai+10], in a formalism close to matrix factorization.

As neural networks, matrix factorization techniques are representation learning methods. However, as we saw in chapter 2, neural networks learn coding and decoding functions, so they can project data easily into the representation space. Matrix factorization does not, it proposes instead to learn dictionaries and codes. Computing the code of unseen example is called the out-of-sample problem and can be treated as regression problem. It is possible to find this code and update the dictionary at the same time as [BG09] does to remove the background of video streams. Note that is is possible to learn coding functions using a framework close to matrix factorization, called projective factorization [YO10], which shows that neural networks and matrix factorization share similarities. This relationship implies that the control complexity mechanisms and supervision [BL+12] or semi-supervision [Wan+09] constraints that we saw for neural networks can be transferred to matrix factorization. However, the benefit of matrix factorization not learning coding function but dictionaries and codes, with the exception of kernel methods such as [Sch+97], is that learned representations are interpretable.

This and their plasticity make matrix factorization techniques reference models for representation learning. They are intensively studied and are state of the art models in such areas as blind source separation and recommender systems. As do neural networks, they benefit from advances in terms of computing capabilities to handle very large matrices [Yin+14]. We provide in this chapter background material on matrix factorization that we formalize as the minimization of a cost function between the data matrix and reconstruction matrix, the product of explanatory factors. We will first consider the low rank approximation problem in such a formalism and its optimal solution, in the sense of the least squares, given by the SVD. We will then present in detail the two classical constraints that are non-negativity and sparseness. We will conclude the chapter by a quick review of various other constraints and formalisms.

3.1 Low-rank approximation

Let us begin by introducing the concept of low-rank approximation. Matrix factorization assumes that observed examples can be explained by a limited number of hidden factors. We will start by explaining how this concept of explaining factors is related to matrix products. As we have seen in section 2.1.2, many learning algorithms can be expressed as the minimization of a loss function that assesses the quality of the model on the training set. Matrix factorization is no exception and the losses quantify the differences between the data matrix and the reconstruction matrix.

3.1.1 Optimization problem

We consider a dataset of m examples described by n continuous features. We represent this dataset by a data matrix that we denote $\mathbf{X} \in \mathbb{R}^{n\times m}$, so each column of $\mathbf{X}$ is an example $\mathbf{x}_i \in \mathbb{R}^n$. We define the problem of matrix factorization as finding two factor matrices $\mathbf{D} \in \mathbb{R}^{n\times k}$ and $\mathbf{H} \in \mathbb{R}^{k\times m}$ such that their product $\hat{\mathbf{X}} = \mathbf{DH}$, the reconstruction matrix, is as close as possible to the data matrix $\mathbf{X}$. This closeness is expressed as a loss: the reconstruction error, that is low when $\hat{\mathbf{X}} = \mathbf{DH}$ is close to $\mathbf{X}$. Solving the factorization is finding the values of $\mathbf{D}$ and $\mathbf{H}$ that minimizes the loss. As for classification, many loss function have been studied. In this thesis we chose to use the mean square error:

$$\mathcal{L}(\mathbf{X}, \mathbf{H}, \mathbf{D}) = \|\mathbf{X} - \hat{\mathbf{X}}\|_{Fro}^2 = \|\mathbf{X} - \mathbf{DH}\|_{Fro}^2 = \sum_{ij} (\mathbf{X}_{ij} - (\mathbf{DH})_{ij})^2 \tag{3.1}$$

We will now proceed to a quick analysis of the dimensions of $\mathbf{D}$ and $\mathbf{H}$ to understand their respective roles in the factorization. $\mathbf{D}$ contains k elements of dimension n (the same as the input space) and $\mathbf{H}$ contains m elements in dimension k. The i-th

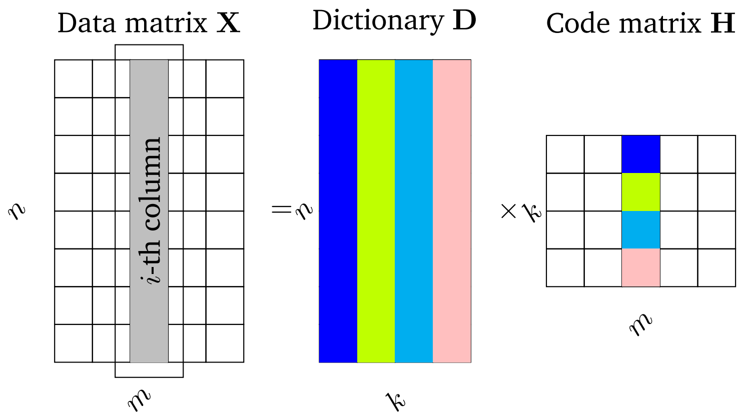


Fig. 3.1.: The i-th column of $\mathbf{X}$, the data matrix, is computed as a linear combination of columns of the dictionary $\mathbf{D}$. The coefficient of this combination are given by the i-th column of the code matrix $\mathbf{H}$.

column of $\mathbf{X}$ (the i-th example of the dataset) is explained by a combination of elements of $\mathbf{D}$ whose factors are stored in the i-th column of $\mathbf{H}$:

$$\mathbf{X} = \mathbf{D}\mathbf{H} \Leftrightarrow \forall i \in \{1, 2, \dots, m\},\ \mathbf{x}_i = \sum_{j=1}^{k} \mathbf{H}_{ji}\mathbf{d}_j \tag{3.2}$$

The same principle is presented in figure 3.1 where the coloring explicits the coefficient used for each element of $\mathbf{D}$ in the combination that computes the i-th row of $\mathbf{X}$. We call D the dictionary matrix as its columns are basic elements, prototypes or atoms, that are used to reconstruct every example of the dataset. H is the hidden representation matrix as each of its columns represents the new coordinates of an example on the dictionary.

3.1.2 Singular values decomposition

Solving the matrix factorization, given a data matrix and a loss, is finding the best approximation of the data matrix as a product of factors. This involves a large number of parameters and, in practice, requires a regularization framework not to be too sensitive to small variations in the data. One possible regularization framework is to limit the rank of the reconstruction $\hat{\mathbf{X}} = \mathbf{D}\mathbf{H}$ to be at most r. The matrix factorization problem under the rank limitation of the reconstruction matrix is called the low rank approximation problem:

$$\min_{\mathbf{H},\mathbf{D}} \mathcal{L}(\mathbf{X}, \mathbf{H}, \mathbf{D}) \text{ s.t. } \operatorname{rank}(\hat{\mathbf{X}}) \leq r \tag{3.3}$$

Singular Value Decomposition provides an answer to the low-rank approximation problem presented in equation (3.3) Let us denote $\sigma \in \mathbb{R}_+$ a nonnegative scalar and $\mathbf{u} \in \mathbb{R}^n$ and $\mathbf{v} \in \mathbb{R}^m$ two vectors. $\mathbf{u}$, $\mathbf{v}$ and σ are respectively called left and right singular vectors and singular value of $\mathbf{X} \in \mathbb{R}^{n \times m}$ if they verify:

$$\mathbf{X}\mathbf{v} = \sigma\mathbf{u} \text{ and } \mathbf{X}^t\mathbf{u} = \sigma\mathbf{v} \tag{3.4}$$

A complete decomposition is a set of such triplets $(\mathbf{u}, \mathbf{v}, \sigma)$ with an additional constraint: the set of left singular vector and the set of right singular vectors should both be orthonormal. Summing up all notations and denoting $\mathbf{U}$ and $\mathbf{V}$ the matrices having left and right singular vectors as columns and Σ a diagonal matrix containing the singular values σ of $\mathbf{X}$ as its diagonal coefficients, the singular value decomposition of $\mathbf{X}$ is a set of matrices $\mathbf{U}, \mathbf{V}, \Sigma$ such that:

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^t,\ \mathbf{U}\mathbf{U}^t = \mathbf{V}\mathbf{V}^t = I \tag{3.5}$$

Such a decomposition always exists, for any real $\mathbf{X}$ matrix[2]. This comes from the fact that $\mathbf{X}\mathbf{X}^t$ is symmetric and positive semi-definite thus can be diagonalized, using the spectral theorem, with leads to the SVD. It also explains why the singular values are positive: they are the eigen-values of the positive semi-definite matrix $\mathbf{X}\mathbf{X}^t$

As singular values are positive by definition, the matrix Σ contains only positive real number on its diagonal (and zero elsewhere). We can then define $\sqrt{(}\Sigma)$ as applying the squared root to all entries of Σ. As the matrix is diagonal, $\sqrt{(}\Sigma)$ satisfies the property $\sqrt{(}\Sigma).\sqrt{(}\Sigma) = \Sigma$. We exploit this property to express the SVD using to factors H and D in equation (3.6).

$$\hat{\mathbf{X}} = \mathbf{U}\Sigma\mathbf{V}^t = \mathbf{D}\mathbf{H},\ \mathbf{D} = \mathbf{U}\sqrt{\Sigma} \text{ and } \mathbf{H} = \sqrt{\Sigma}\mathbf{V}^t \tag{3.6}$$

3.1.3 Truncated Singular Value Decomposition

In the solution provided by equation (3.6), the rank of the reconstruction matrix $\hat{\mathbf{X}}$ is a free parameter. Limiting the rank of $\hat{\mathbf{X}}$, with this formulation, is exactly limiting the number of singular values to use. We denote $\Sigma^{(r)} \in \mathbb{R}^{r \times r}$ the diagonal matrix only the r largest singular values sorted in non-increasing order, $\mathbf{U}^{(r)} \in \mathbb{R}^{n \times r}$ and $\mathbf{V}^{(r)} \in \mathbb{R}^{m \times r}$ the matrices storing the r corresponding left and right (respectively) singular vectors. The truncated SVD ($\Sigma^{(r)}$, $\mathbf{U}^{(r)}$, $\mathbf{V}^{(r)}$) is computed as:

$$\mathbf{X} \approx \hat{\mathbf{X}} = \mathbf{D}\mathbf{H},\ \mathbf{D} = \mathbf{U}^{(r)}\sqrt{\Sigma^{(r)}} \text{ and } \mathbf{H} = \sqrt{\Sigma^{(r)}}\mathbf{V}^t_{(r)} \tag{3.7}$$

[2]It also works for complex matrices.

According to the Eckart-Young theorem, the best reconstruction matrix $\hat{\mathbf{X}}$ of rank r, in the sense of the mean squared error, of any given matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ is given by the truncated SVD ($\Sigma^{(r)}$, $\mathbf{U}^{(r)}$, $\mathbf{V}^{(r)}$) that we have just defined.

3.2 Constrained matrix factorization

In practice there is a link between the rank r of the reconstruction matrix $\hat{\mathbf{X}}$ and the inner dimension k of the product. The latter is an upper-bound of the former. Thus, for other regularization framework that do not directly constrain the rank of $\hat{\mathbf{X}}$, choosing small values of k still act as a regularization. We will focus here on the non-negative and sparse matrix factorization that are two popular regularization framework for matrix factorization that do not directly constrain the rank of $\hat{\mathbf{X}}$ but rather impose the nature of the factors.

3.2.1 Non-negative matrix factorization

Images, texts or ratings have natural representations that are non-negative and this non-negativity is meaningful and interpretable as concepts and proportions for the dictionary and the code matrix. Elements of the dictionary are concepts respecting the non-negativity of the data that are shared up to a certain proportion by each example and this proportion is given by the code of each example. Consider texts represented as binary bag of words: each dimension indicates the presence (1) or absence (0) of a word in the text. As we have seen, $\mathbf{D}$ amounts to set of concept that are common in the input data and use them to compute the new representation. The matrix $\mathbf{H}$ holds the coefficients of the new representation and of the combination of rows of $\mathbf{D}$ that reconstruct each input example. Using traditional techniques, like the truncated SVD, does not guaranty the sign of entries of $\mathbf{D}$ and $\mathbf{H}$. For our text example, it leads to two annoying facts. Firstly, we can end up with concepts, columns of $\mathbf{D}$, having negative values that we cannot interpret. Respecting the natural domain of the input data would be more sensible. Secondly, linear combinations with negative values are hard to grasp. It often leads to complex situations where concepts cancel each others and are hard to fully understand. Non-negative matrix factorization was introduced to respect the non-negativity of input data and to obtain interpretable decompositions. It replaces the direct constraint on the rank of $\hat{\mathbf{X}} = \mathbf{H}.\mathbf{D}$ by non-negativity constraints:

$$\min_{\mathbf{H},\mathbf{D}} \mathcal{L}(\mathbf{X}, \mathbf{DH}) \text{ s.t. } \mathbf{H} \geq 0 \text{ and } \mathbf{D} \geq 0 \tag{3.8}$$

The non-negative constraint forces the models to reconstruct each data point using only positive combinations of positive basic elements, the columns of $\mathbf{D}$. It acts as a strong regularization of the parameters but also leads to part-based representations: as described in [LS99], it forces the extraction of small parts occurring frequently in the training set and combine them together to reconstruct each data point.

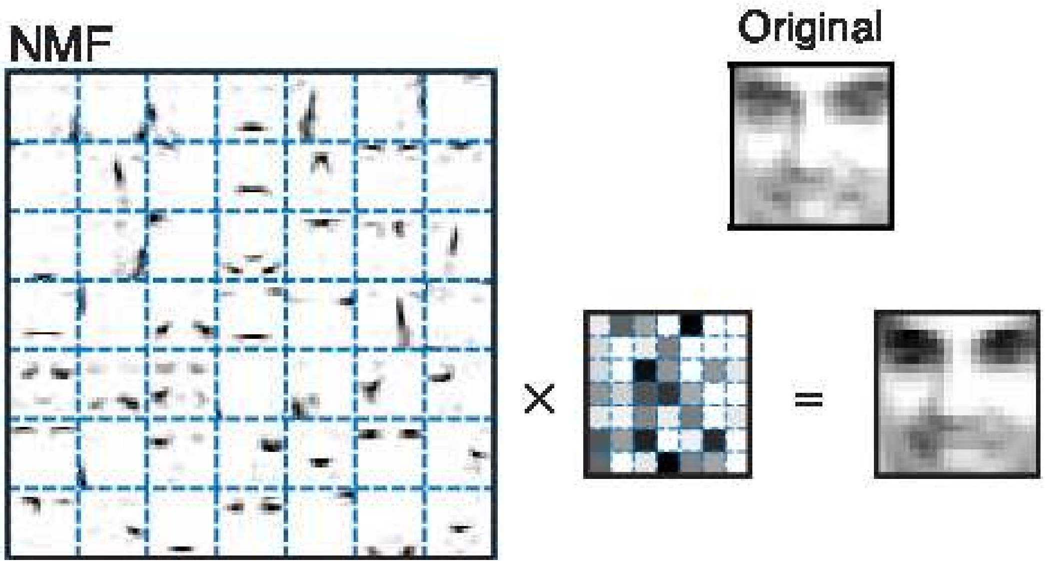


Fig. 3.2.: Illustration of part-based representations extracted by NMF, on a face recognition task, from [LS99]. The matrix on the left of the $\times$ sign is the dictionary $\mathbf{D}$ that contains the basic elements. The reconstruction is done here for one image only $\mathbf{x}$, that is represented mathematically as a vector for the model but represented visually as a matrix. The vector is the flatten version of the matrix. The matrix just right to the $\times$ sign is the code matrix $\mathbf{h}$ that contains the contribution of each element to the reconstruction vector $\hat{\mathbf{x}}$ of one example only. This reconstruction vector $\hat{\mathbf{x}}$ is presented after the $=$ sign. The original vector $\mathbf{x}$ is presented above.

Non-negative matrix factorization has been used for many applications. In [Xu+03], the authors compare SVD and NMF for document clustering: the H matrix is interpreted as a soft clustering allocation and rows of D as centroids. The work in [Par08] proposes to use NMF to select relevant texts to generate summaries. The part-based representations extracted by NMF work well with image analysis tasks like facial recognition [Zaf+06]. It has also been applied to pitch [BL+12] and musical instrument [Ben+06] classification.

The popularity of Non-negative Matrix Factorization (NMF) has largely benefited from the algorithm proposed by [LS01]. The problem is not convex in $\mathbf{H}, \mathbf{D}$ but is in $\mathbf{H}$ when $\mathbf{D}$ is fixed and vice versa. The solution proposed in [LS01] exploits this through alternate updates of $\mathbf{H}$ and $\mathbf{D}$: $\mathbf{H}_{p+1}$ is computed first using $\mathbf{H}_p$ and considering $\mathbf{D}$ and $\mathbf{X}$ fixed, the same is then done for $\mathbf{D}$:

$$\mathbf{H}_{p+1} = \mathbf{H}_p \odot \frac{\mathbf{D}^t\mathbf{X}}{\mathbf{D}^t\mathbf{D}\mathbf{H}_p} \quad \text{and} \quad \mathbf{D}_{p+1} = \mathbf{D}_p \odot \frac{\mathbf{X}\mathbf{H}^t}{\mathbf{D}_p\mathbf{H}\mathbf{H}^t} \tag{3.9}$$

These update rules are derived smartly fixing the learning rate of a gradient descent and are proved to converge in [LS01]. The learning rate γ is different for each entry of the matrix:

$$\mathbf{H}_{ij} \leftarrow \mathbf{H}_{ij} - \gamma_{ij}[\mathbf{D}^t(\mathbf{X} - \mathbf{D}\mathbf{H})]_{ij} \text{ s.t. } \gamma_{ij} = \frac{\mathbf{H}_{ij}}{[\mathbf{D}^t\mathbf{D}\mathbf{H}]_{ij}} \tag{3.10}$$

A similar derivation is done for the updates of $\mathbf{D}$. The associated algorithm is presented in algorithm 4.

```
Data: $\mathbf{X}$, $\mathbf{H}_0$, $\mathbf{D}_0$
Result: $\mathbf{H}$, $\mathbf{D}$
$\mathbf{H} \leftarrow \mathbf{H}_0,\ \mathbf{D} \leftarrow \mathbf{D}_0$;
while convergence is not reached do
    $\mathbf{H} \leftarrow \mathbf{H} \odot \mathbf{D}^t\mathbf{X} \oslash \mathbf{D}^t\mathbf{D}\mathbf{H}$;
    $\mathbf{D} \leftarrow \mathbf{D} \odot \mathbf{X}\mathbf{H}^t \oslash \mathbf{D}\mathbf{H}\mathbf{H}^t$;
end
```

Algorithm 4: Multiplicative update rules for NMF using Frobenius loss.

The algorithm proposed by [LS01] is simple to implement and fast enough, in practice, for big matrices, assuming that everything fits in memory. One of the great advantage of this algorithms is the absence of learning rate to tune as it is smartly fixed to derive the multiplicative rules. However, it was argued in [Lin07] that projected gradient methods with correct learning rate search techniques converge faster to a good solution than these multiplicative update rules.

3.2.2 Sparse Matrix factorization

Sparse constraints provide an other interpretation of *simple* factors. Instead of limiting the rank of the reconstruction matrix, they limit the number of non-zero entries. In the context of matrix factorization, these constraints are inherited from other domains, such as representation learning and dictionary learning where sparseness is widely used. As described in [Lee+06], high sparseness means a compressed representation even though the number of dimensions of the hidden space still has many dimensions as few of them are active for each individual example. Replacing the rank constraint in (3.3) by a sparseness constraint gives the formulation in equation (3.11).

$$\min_{\mathbf{H},\mathbf{D}} \mathcal{L}(\mathbf{X}, \mathbf{D}, \mathbf{H}) + \lambda_{\mathbf{H}}\|\mathbf{H}\|_1 \text{ s.t. } \forall i, \|\mathbf{d}_i\|_2 = 1 \tag{3.11}$$

In equation (3.11), $\|\mathbf{H}\|_0$,the number of non-zero entries in the matrix $\mathbf{H}$, should be used instead of $\|\mathbf{H}\|_1$, the sum of absolute values of entries of $\mathbf{H}$. However, it is impossible to derivate $\|\mathbf{H}\|_0$ and $\|\mathbf{H}\|_1$ is often used as a surrogate. The constraints on the columns $\mathbf{d}_i$ of $\mathbf{D}$, that they must have unit Euclidean norm, is there to discard all the solutions that can be obtained by multiplying $\mathbf{D}$ by any real number greater than one and dividing $\mathbf{H}$ by the same number.

Stochastic updates

Multiplicative update rules are batch rules. It is possible to use stochastic (or mini-batch rules) as well. However, in practice, it is necessary to control the evolution of the dictionary $\mathbf{D}$ so it is not drastically modified after each stochastic update. One solution comes from the domain of dictionary learning [Mai+10] by using incremental updates and memory terms. These memory terms balance the influence of the new data used to update the parameters and the previously seen examples. [Mai+10] uses a derived loss function, expressed for only one example $\mathbf{x}$:

$$\min_{\mathbf{H},\mathbf{D}} \mathcal{L}(\mathbf{x}, \mathbf{D}.\mathbf{h}) + \lambda_{\mathbf{H}} \|\mathbf{h}\|_1 \text{ s.t. } \forall i, \|\mathbf{d}\|_2 = 1 \tag{3.12}$$

The algorithm developed by [Mai+10] comes from dictionary learning and focuses on learning $\mathbf{D}$. For each update, an example $\mathbf{x}_i$ is drawn and its sparse coding $\mathbf{h}_i$ is computed using the Lasso [Tib96] considering $\mathbf{D}$ fixed. The two memory parameters are updated to take into account the input vector $\mathbf{x}_i$ and its representation $\mathbf{h}_i$, as well as previously seen examples. The update of the dictionary $\mathbf{D}$ is then done using these memory parameters. Also, in practice, the constraint used on the columns of $\mathbf{D}$ is relaxed to $\mathbf{d}_i.\mathbf{d}_i \leq 1$ to avoid increasing the magnitude of columns whose norm is already lower than 1.

In [BG09], the NMF is adapted to handle time-evolving datasets like video streams. The hypothesis here is that, after a while, the prototypes in $\mathbf{D}$ should be relatively stable when updated to take into account a new example in $\mathbf{X}$. This leads the authors to incremental updates of the parameters when each new image is received and is closely related to updates of [Mai+10]. However, contrary to [Mai+10], they proposed multiplicative update rules that are derived as explained in 3.2.1. They also introduce memory terms and associated weights to balance the forgetfulness of the parameters.

Stochastic updates are also useful when the data matrix $\mathbf{X}$ is not fully observed. It provides a simple way to update the parameters using only the observed value and is the basic method for recommender systems [Kor+09] when gradient updates require the use of a filter to avoid considering unobserved parts of the matrix. It is part of the relational learning framework. The work of [SG08] describes how to use stochastic gradient descent in this context: parameters $\mathbf{h}$ and $\mathbf{D}$ are learned from the observed values of $\mathbf{X}$ and used to complete the missing values. More details are given in chapter 4.

The main benefit of stochastic [Mai+10; SG08] and incremental [BG09] updates is scalability to large datasets. In [Mit+10], the authors propose another approach to large-scale factorization using low-rank semidefinite programming: they use a convex relaxation of the non-convex reconstruction loss of (3.2) and a quasi-Newton solver. The recent [Yin+14] decomposes the multiplicative update rules of [LS01] to scale to very large matrices using large computer clusters.

3.2.3 Sparse Non-negative Matrix Factorization

It is of course possible to adapt non-negative matrix factorization by adding sparseness constraints. The reference algorithm was proposed in [Hoy02]. The sparseness constraint on $\mathbf{H}$ can easily be incorporated into the multiplicative update rules. However, it is not the case for the constraint of unit-norm rows of $\mathbf{D}$. [Hoy02] proposes to use projected gradient instead of the multiplicative update rules for the dictionary $\mathbf{D}$, as presented in algorithm 5.

```
Data: $\mathbf{X},\ \mathbf{H}_0,\ \mathbf{D}_0,\ \lambda_{\mathbf{H}},\ \mu$
Result: $\mathbf{H},\ \mathbf{D}$
$\mathbf{H} \leftarrow \mathbf{H}_0,\ \mathbf{D} \leftarrow \mathbf{D}_0$;
while convergence is not reached do
    $\mathbf{H} \leftarrow \mathbf{H} \odot \mathbf{D}^t\mathbf{X} \oslash (\lambda + \mathbf{D}\mathbf{H}\mathbf{H}^t)$;
    $\mathbf{D} \leftarrow \max(0, \mathbf{D} + \mu(\mathbf{X} - \mathbf{D}\mathbf{H})\mathbf{H}^t)$;
    Normalize rows of $\mathbf{D}$;
end
```

Algorithm 5: Learning Sparse NMF from [Hoy02].

As in algorithm 4, $\odot$ and $\oslash$ are the element-wise multiplication and division on matrices. This algorithm has more parameters to tune: μ the learning rate for the gradient descent on $\mathbf{D}$ and $\lambda_{\mathbf{H}}$ the weight of the sparseness constraint on $\mathbf{H}$.

Matrices have two dimensions, rows and columns, thus the sparseness can be interpreted in many way. Both [Mai+10] and [Hoy02] propose the same vision of sparseness: a few non-zero entries overall in $\mathbf{H}$. The interesting work in [Hoy04] provide four interpretations of sparseness.

Sparse rows of $\mathbf{H}$ means that each concept, each column of $\mathbf{D}$, should be used to reconstruct only a small number of examples, columns of $\mathbf{X}$. Concepts of $\mathbf{D}$ are then expected to be more discriminant.

Sparse rows of $\mathbf{D}$ means that each input feature should be used in as few as possible concepts, columns of $\mathbf{D}$. Learned parts are expected to be more discriminant.

Sparse columns of $\mathbf{H}$ means that each example, each column of $\mathbf{X}$, is constructed using only a few elements of $\mathbf{D}$. It is the classic vision of sparseness.

Sparse columns of $\mathbf{D}$ means that each concept, each column of $\mathbf{D}$, should use as few features as possible. As argued above, NMF extract part-based representations and this aims at extracting small parts.

3.3 Extensions of the Matrix Factorizations

The popularity of matrix factorization depends on the plasticity of its original problem formulated in equation (3.3). This problem can easily be extended by adding more constraints on the parameters or slightly modifying the formulation. We propose a brief review of these extensions here. However the list is of course not exhaustive.

3.3.1 Various Non-negative Matrix Factorization

As discussed in 3.2, the NMF only adds non-negative constraints to the problem formulation. It is still extremely generic and variations of NMF have been extensively studied.

In the context of using NMF as a clustering method, which views $\mathbf{D}$ as centroids and $\mathbf{H}$ as cluster allocations (soft or hard), [LD06] describes many possible variations of NMF. For instance, semi-non-negative matrix factorization [Din+06] preserves the property of only positive combination of atoms on negative data by removing the non-negativity constraint on $\mathbf{D}$ but keeping it on $\mathbf{H}$. A convex NMF is also presented. It works on a data matrix $\mathbf{X}$ that is not necessarily non-negative and find $\mathbf{H}$ and $\mathbf{D}$, non-negative, such that $\mathbf{X} \approx \mathbf{DHX}$. This formulation is close to the one of autoe-encoders that we have seen in chapter 2. Manifold or graph regularizations [Cai+08; Cai+09; Cai+11] can force a geometric structure in the extracted representation. It is also possible to use more than two factors $\mathbf{H}$ and $\mathbf{D}$ to explain the data matrix $\mathbf{X}$ as in [Li+09]. For instance, in the context of tri-factorization, another factor $\mathbf{S}$ is added to factorize $\mathbf{X}$ as $\mathbf{DSH}$. This $\mathbf{S}$ can be used to group concepts, elements of the dictionary, together to increase the level of abstraction of the representation: the code matrix operates here on groups of concepts and not concepts directly.

3.3.2 Supervised Factorizations

If the dataset is supervised, it is possible to use this supervision to guide the learning of $\mathbf{H}$ and $\mathbf{D}$, as presented in [PZ11] for a face recognition application. It is then expected that prototypes in $\mathbf{D}$ are discriminant. Similarly, but for pitch identification [BL+12] uses a supervision constraint during the learning procedure. Moreover, the NMF is compared to a similar approach using auto-encoders, once again showing the close relationship between the two families of models. In [DGX11], a maximum margin criterion, as in Support Vector Machines (SVM), is added to the reconstruction error of (3.8). A study of the links between SVM and NMF is proposed in [Pot11]. An interesting idea is developed in [Wan+13] where each atom of the dictionary $\mathbf{D}$ is expected to be the parameter of a linear classifier.

The central idea to supervised matrix factorization is to impose that the code matrix $\mathbf{H}$ leads to good classification performance. The most common approach is to add another term to the loss function, a classification term using $\mathbf{H}$ and a given model. For instance, [PZ11] integrates a term corresponding to learning a k-nearest neighbor classifier on $\mathbf{H}$ while learning the factorization. Other approaches have been proposed. In [Jam+10], the authors introduce a criterion based on term frequency and inverse document frequency (TF-IDF) on the code matrix $\mathbf{H}$ to separate the classes and not a classifier on $\mathbf{H}$. Clustering approaches maximizing the similarity inside each class and the distance between separate classes, has also been proposed for NMF in [Kot+07]. [Lia+10] extends this approach using kernels. Finally, another idea, introduced in [Mai+08], is to learn one dictionary $\mathbf{D}_c$ per class c and to use the residual $r_c = \|\mathbf{x} - \mathbf{D}_c\mathbf{h}_c\|^2$ for classification. The idea is that elements of the class c should be best reconstructed by the dictionary $\mathbf{D}_c$ learned on examples coming from the class c. The principle of extracting one dictionary per class is exploited in [Wan+12] to speed up learning and scale up matrix factorization techniques.

3.3.3 Semi-supervised Factorizations

As labels are usually expensive, datasets are often only partially labeled. Semi-supervised models have been developed to deal with this situation exploiting both labeled and unlabeled training examples, they are covered in details in [Cha+06]. Non-negative matrix factorization can also benefit from the semi-supervised framework. A first approach is to factorize the similarity matrix of the dataset instead of the data matrix as presented in [Liu+06]. In [Wan+09], the similarity between labeled examples is exploited so that points similar in the input space must lie in similar regions of the latent space, using techniques similar to [Cai+09]. [LW10] adds the constraints that examples with the same label must be projected at the same point in the latent space while unlabeled examples are unconstrained. Another approach developed in [CS11] forces the geometry of the latent space to respect the margins of the input space. Two semi-supervised support vector machines are learned, the former in the input space and the latter in the latent space. As the parameter vector of a SVM can be expressed as a linear combination of training examples, [CS11] constrains the parameter vectors of both models (input and latent spaces) to share the same coefficient in the linear combination. In the context of the tri-factorization proposed in [Li+09], two factors are explicitly linked to supervision and must stay close, for labeled examples, to the actual value given by the supervision.

3.3.4 Probabilistic models and factor analysis

The formulation that was presented above is deterministic. We briefly review here work on corresponding probabilistic models. They can be seen as extending factor analysis, models aiming at explaining the variability of observed data points $\mathbf{x}$ using latent variables $\mathbf{h}$. A classic formulation of factor analysis is presented in (3.13) where $\mathbf{D}$ is a linear projector, μ accounts for the mean, centering the data, and ϵ is a random noise.

$$\mathbf{x} = \mathbf{D}\mathbf{h} + \mu + \epsilon \tag{3.13}$$

Probabilistic Principal Component Analysis, introduced in [TB99], builds directly on this by modeling the conditional probability $\mathbf{x}|\mathbf{h}$ of $\mathbf{x}$ given $\mathbf{h}$ as following a Gaussian distribution of mean $\mathbf{D}\mathbf{h} + \mu$, as in (3.14). This derivation was made considering the random noise ϵ to be a Gaussian white noise of variance σ^2: $\mathcal{N}(0, \sigma^2\mathbf{I})$.

$$\mathbf{x}|\mathbf{h} \approx \mathcal{N}(\mathbf{D}\mathbf{h} + \mu, \sigma^2\mathbf{I}) \tag{3.14}$$

For matrix factorization [Kor+09], factor models have been adapted to explain ratings with a similar formulation. In the context of recommendation of user comments, [Aga+11] introduces latent vectors to account for ratings and rater/comment affinity. The recent paper [Xu+13], introduces a probabilist context to the deterministic max-margin models [DGX11] and a procedure to automatically select the number of latent dimensions.

3.4 Conclusion

Matrix factorization will be used in all our contributions. In chapters 4 and 5 where they are reference methods for collaborative filtering and recommender systems. In chapter 6 to extract a latent representation that strongly characterizes users based on activity logs, in the context of urban mobility. Interestingly, even though we chose a different formalism, our applications can be formalized as blind source method problems, a field where matrix factorization techniques are reference models since ICA [Com94]. As said in the introduction, for recommender systems the observed signal is the set of user reviews and the hidden sources are the latent representations of users and items. For the extraction of temporal activities out of user traces in chapter 6, the signal is the trace, the hidden sources are the activities of users.

In this chapter, we have provided background material on matrix factorization. Our intention clearly was to differentiate them from neural networks. We believe that matrix factorization techniques are under-rated. They are excellent data mining tools, mainly because they belong to a legacy of explanatory models [Hot33; GVL12; Com94]. Still, they managed to survive over time, thanks to their plasticity and the various constraints that can be added to control the bias/variance trade-off. The most common regularizations are the non-negative [LS01] and sparse constraints [Hoy02]. Matrix factorization is still a field of intense research. One of the current challenges is how to scale to larger datasets, whether they come from video streams [BG09] or recommender systems [Yin+14].

Extending recommender systems 4

Contents

One of the long term goal of machine learning is the design of context aware models. Today, thanks to the availability of data, many models learn the preferences of users to provide personalized access to information. This domain of preference learning [FH10] is quite recent and is one step toward this long term goal. However, these user-centered models are ubiquitous on the Internet today and many people benefit from services using them on a daily basis. Search engines rerank the results of users' queries with respect to the individual preferences of users [Tee+05], based on their browsing history. On-line dating website use user profiles to recommend matching profiles to their subscribers [ML10; Dia+10]. Finally, merchant websites extract user profiles out of navigation history and user reviews to recommend items to their visitors [Sch+99]. We consider the latter context, the recommendation of items, which is the most studied in the literature, due to the availability of user reviews on the Web 2.0. These are traces of opinions expressed by users on goods or services, and consist of four elements:

User the writer of the review. In most cases, users are forced to login to write reviews. The latter are therefore associated with a user and it is possible to track the reviews of each user over time. These reviews are used to build user profiles reflecting the taste and expectations of user.

Item The good or service described by the user. It can be an item on a commercial website, Amazon offers goods ranging from books to microwave ovens, a type of product analyzed by community of amateurs like beers on RateBeer, or a service such as restaurants on Yelp!. They are part of the catalog of the website and the challenge is to recommend relevant items to each user.

Rating It summarizes the overall assessment of an item by a user. It depends both on the user (objective, biased, severe ...) and on the product (quality, popularity, marketing ...). It is traditionally represented by an integer rating from 1 to 5 (stars). The higher it is, the more the user likes the item.

text The text describes in details the rating and the opinion of the user on the item. Our hypothesis is that considering the text, a rich medium to communicate feelings and opinions, enriches user profiles extracted by recommender systems and may improve the relevance of the recommendations. In the text, we look both for opinionated words such as *awesome* or *disgusting* and for traces of the user writing style and consider that both enrich the profiles of users.

The problem of recommendation can be defined in to main ways: the prediction of ratings or the generation of ranked lists of items. The first estimates the rating a user gives to an item (thus modeling his interest in the latter). This is a regression setup and the quality of the prediction is evaluated with respect to its distance to the actual rating (eg using least squares error). This is the paradigm used by the GroupLens engine [Res+94] and for the Netflix challenge [BL07; Kor08] and it is the approach that we have adopted. The second ranks items based on user preferences and returns only the top k (k is hand-fixed) most relevant items [Bre+98; MH04]. This approach is used on e-commerce sites, which recommend a fixed number of items to visitors on each page of the website and the model is then evaluated using TopK measures (recall and precision) or area under the ROC curve.

Four main approaches to recommender systems are generally distinguished in the literature: content based, knowledge based, collaborative filtering and models hybridizing the former ones. Several possibilities of hybridization between different systems are presented in [Bur02]. Content based methods use navigation history (past reviews) and item descriptors [PB07], such as keywords [AT05]. These methods allow an easy integration of metadata. However it is impossible to provide a personalized recommendation to a new user and these models are intrinsically dependent on the quality of the descriptors provided, if any. Knowledge-based methods generally use prior knowledge on the item to recommend (house, car, financial product) [Bur00]. This prior knowledge is used to establish scenarios that lead to a recommendation when no history is available for the user. They provide a solution to the troublesome problem of cold start: presenting the user a personalized recommendation when no or little history is available. Both content based and collaborative filtering suffer from this problem. In this work, we focus on the enrichment of user profiles and have not looked specifically at the cold start problem. However, enriching profiles using text information is known to help in this matter [ML13a].

We tackle problems with large catalogs of items for which we do not always have descriptions or metadata but for which we do have many user reviews. Our work fits naturally in the context of collaborative filtering, which is based solely on past reviews. This framework has been proposed for the first time with the GroupLens engine [Res+94]. The approach is further divided into two sub-categories: memory based and model based approaches. The former is based on k nearest neighbors algorithms [Kor08]. For the latter, prediction models are trained on user review datasets [BL07]. The most common algorithms are matrix factorization and RBM (Restricted Boltzmann Machine) [Kor08; KB11]. Matrix factorization simultaneously extracts users and items profiles [Kor+09]. Moreover, it allows the consideration of bias models (user, item and overall bias) to reflect the characteristics of the users, items and the overall data and simple integration of additional constraints. For these reasons, we have chosen this approach. Our contributions propose to consider in addition to profiles extracted from ratings, profiles extracted from the texts of the reviews with notions coming from sentiment analysis.

Our contributions can be considered as methods to extract robust descriptors of items from texts written by the users that enrich the profiles learned by matrix factorization. Similar approaches have been proposed in [Gan+09; ML13a], confirming the interest of the text part of user reviews. In [Gan+09], the polarity of all sentences of the text of a review are averaged to predict a rating. This confirms the interest of sentiment based approaches to consider the text of reviews but require a hand labeling of sentences of the training reviews. Our models do also consider the polarity of texts while extracting textual profiles. Contrary to [Gan+09], our models do not require sentences to be labeled and thus scale easily to large datasets. In [ML13a], latent profiles are extracted from the texts, and the authors impose a link between latent textual profiles and latent rating profiles. The former are extracted with LDA (Latent Dirichlet Allocation), the latter are computed with matrix factorization. The training algorithm alternates the optimization of the LDA and the matrix factorization which is quite cumbersome. Moreover they discard the polarity of texts in the process while our models preserve this information.

We propose two contributions in the field of recommender systems. Both are based on the assumption that the text is an important way to share feelings and opinions and that the writing style of users is a relevant part of their profiles. Our first contribution is the enrichment of profiles extracted from ratings. We show that raw and latent text profiles provide similar performance for the rating prediction task but at different computational costs. Additionally, considering the polarity of texts helps predicting more accurate ratings. While recommender systems are traditionally confined to correctly predict ratings, our second contribution is to extend this by also predicting the text of a review. Of course, the generation of natural text is out of reach and we formalize the problem as a summarization problem. For a given user and item couple, we select among all texts written by other users on the item, the sentences that are the most likely to express the opinion of the user on the item. To the best of our knowledge, we are the first to propose such a task. We also introduce an evaluation framework based on the ROUGE metric [Lin04].

This chapter is organized as follows. We will first define classical models of collaborative filtering as well as review current advances in recommender systems. We will then present our first contribution introducing text profiles and considering the polarity of these text as an additional information for rating prediction. Finally, we detail our second contribution: generating a personalized summary for each user in addition to the rating prediction.

4.1 Related work

As discussed in the introduction of this chapter, we place ourselves in the context of collaborative filtering for recommender systems. We propose to refine the user and item profiles by considering information extracted out of the review texts and to generate predictive summaries that are as close as possible to the actual review texts. We start this section by presenting the classic models of collaborative filtering that we use as building blocks. Then, we provide background material on summarization, introducing the concepts used for the generation of our predictive summaries.

4.1.1 Collaborative filtering recommender systems

Collaborative filtering recommender systems classically use matrix factorization to extract latent profiles for users and items from the rating matrix. This matrix $\mathbf{R} \in \mathbb{R}^{n_U \times n_I}$ is built by indexing the n_U users and n_I items from the reviews and setting the value of each entry $\mathbf{R}_{ui}$ to the value of the rating r_{ui} of the review $(u, i, r_{ui}, \mathbf{d}_{ui})$. We use matrix factorization in this work, with additional bias terms. We present these bias terms first, and then the matrix factorization.

Overall bias

The first (and simplest) bias term is the overall bias. It is a constant model that always predicts the average rating computed on the training set:

$$g_0 = \frac{1}{m} \sum_{(u,i) \in Train} r_{ui} \tag{4.1}$$

The overall bias gives a first indication of the rating habits in the dataset. It is however a bit dull as it considers identically all users and items.

User bias

The second bias term g_1 is the user bias. It assumes that each user has a different rating behavior and rates all items similarly. The prediction for review (u, i) will be the average of the m_u ratings given by u in the training set (denoted $Train$) and is independent of the item i:

$$g_1(u) = \frac{1}{m_u} \sum_{(u,i') \in Train} r_{ui'} \tag{4.2}$$

The user bias is less blunt than the overall bias but is still too restrictive. Usually, the variance in ratings from one user is high as a user will rate items he likes and items he dislikes. Thus considering the average of user ratings is usually not accurate enough.

Item bias

The third bias term g_2 is the item bias and is the mirror of the user bias, for items. It assumes that each item as an average quality that is acknowledged by all users. The prediction for review (u, i) will be the average of the m_i ratings on item i in the training set and is independent of the user u:

$$g_2(i) = \frac{1}{m_i} \sum_{(u',i) \in Train} r_{u'i} \tag{4.3}$$

Sadly for our free will, even though it does not consider the user, this models is a strong baseline. Communities often agree on quality standards and tend to rate items similarly.

Matrix factorization

Collaborative filtering proposes to go beyond simple averages and to learn user and item profiles as latent factors. This is commonly done using matrix factorization [Kor+09] on the rating matrix:

$$\mathbf{R} \approx \mathbf{\Gamma}_U \mathbf{\Gamma}_I^t \tag{4.4}$$

The two factors are the latent profiles of the user, $\mathbf{\Gamma}_U \in \mathbb{R}^{n_U \times k}$, and item, $\mathbf{\Gamma}_I \in \mathbb{R}^{n_I \times k}$ and $k$is the number of latent variables. Some entries of $\mathbf{R}$ are not defined: these of unobserved ratings. However, we assume that if the training set is large enough, then the number of observed ratings will be enough to learn robust users and items profiles. Then, the value of $\mathbf{\Gamma}_U \mathbf{\Gamma}_I^t$ at the entry (u, i) should be a correct prediction of the rating of the review (u, i). We denote g_3 the matrix factorization prediction function, which is computed as follows[1]:

$$g_3(u, i) =< \gamma_u, \gamma_i >= \gamma_u . \gamma_i \tag{4.5}$$

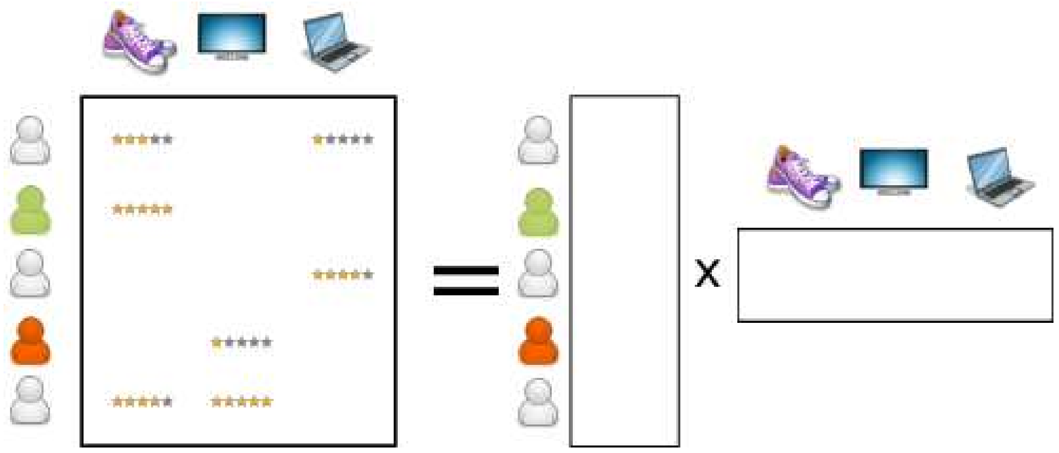


Fig. 4.1.: Ratings given by users are gather in a tabular form, the rating matrix. Each row corresponds to the ratings of one individual user. Each column corresponds to the ratings given to one individual item. This matrix is decomposed as the product of two factors. The left factor is the latent profiles of users, each row is the latent profile of one individual user. The right factor is the latent profiles of items, each column is the latent profile of one individual item.

Factorizing a partially observed matrix

The main difference with all matrix factorization algorithms presented in chapter 3 is that the data matrix here is the rating matrix and that it is not fully observed. The first problem is to adapt the loss to minimize. We stick with the mean squared error

[1] $<, >$ and . are two possible writings for the dot product and γ is a row of $\mathbf{\Gamma}$.

as it fits the regression context for predicting a note in range 1 to 5. The classic formulation of the loss $\mathcal{L}$ would be:

$$\mathcal{L} = \frac{1}{n_u n_i} \|\mathbf{R} - \mathbf{\Gamma_U} \mathbf{\Gamma_I}^t\|_F^2 \tag{4.6}$$

However, this formulation takes into account unobserved values of $\mathbf{R}$ that are not defined. One simple solution to avoid that is to use a mask matrix $\mathbf{M} \in \mathbb{R}^{n_U \times n_I}$ such that its entry (u, i) is 1 if (u, i) is an observed review, 0 otherwise. This formulation is derived in equation (4.7) where $\odot$ is the element-wise matrix multiplication.

$$\mathcal{L} = \frac{1}{m} \|\mathbf{M} \odot (\mathbf{R} - \mathbf{\Gamma_U} \mathbf{\Gamma_I}^t)\|_F^2 + \lambda_U \|\Gamma_U\|_F^2 + \lambda_I \|\Gamma_I\|_F^2 \tag{4.7}$$

$$\mathcal{L} = \frac{1}{m} \sum_{(u,i) \in \ Train} (\gamma_u . \gamma_i - r_{ui})^2 + \lambda_U \|\gamma_u\|_F^2 + \lambda_I \|\gamma_i\|_F^2 \tag{4.8}$$

The latter formulation is commonly preferred as it allows a straightforward implementation of a stochastic gradient descent to minimize the loss. Moreover, instead of storing a (sparse) rating matrix of dimension n_u times n_i it only requires a list of m triplets (r_{ui}, u, i). It is then easy in practice to shuffle the training set and iterate through it. The stochastic gradient descent (SGD) algorithms will draw one triplet (r_{ui}, u, i) and update parameters γ_u and γ_i with respect to it. Please note that to avoid over-fitting, we use $L2$ regularization on both latent representation. Also, it is possible and common to impose non-negativity of factor matrices. Algorithm 6 presents SGD to optimize the loss of equation (4.8) with $L2$ penalties and non-negativity constraint It offers another way to adapt gradient descent to solve non-negative matrix factorization, here from a data matrix that is not fully observed. Contrary to the multiplicative update rules (algorithm 4) and to the projected gradient (algorithm 5), algorithm 6 is stochastic and easily scales to big matrices [2]. There is also a practical difference between stochastic and batch algorithms. In batch algorithms, parameters Γ_U and Γ_I were updated alternatively. Here they are updated as the same time: to make this explicit in algorithm 6, we introduced temporary variables γ'.

To lighten the notations and focus on the gradient descent, algorithm 6 is written using a simple stopping criteria: a fixed number of iterations. This however may not be optimal in practice, as discussed in details in many books such as [Mon+12]. For our experiments, described in details below, we have divided the reviews in training, validation and test sets. As so, we can exploit the loss on the validation set and use an early stopping criterion in addition to a maximal number of iterations: we stop the training if the loss on the validation set stops decreasing. Moreover,

[2]Multiplicative update rules can be modified to scale up as in [Yin+14]

```
Data: $\mathcal{T} = \{(r_{ui}, u, i)\}$, $\Gamma_{U0}$, $\Gamma_{I0}$, $\lambda$, $\mu$, $p$
Result: $\Gamma_U$, $\Gamma_I$
$\Gamma_U,\ \Gamma_I \leftarrow \Gamma_{U0},\ \Gamma_{I0}$;
for $t \in \{1, 2, \ldots, p\}$ do
    Shuffle $\mathcal{T}$;
    for $(r_{ui}, u, i) \in \mathcal{T}$ do
        $\gamma_u$ is the $u$-th row of $\Gamma_U$, $\gamma_i$ is the $i$-th row of $\Gamma_I$;
        Compute prediction $\hat{r}_{ui} = \gamma_u.\gamma_i$ and $\delta \leftarrow \hat{r}_{ui} - r_{ui}$;
        $\gamma'_u \leftarrow \max(0, \gamma_u - \mu.(\lambda\gamma_u + \delta\gamma_i))$;
        $\gamma'_i \leftarrow \max(0, \gamma_i - \mu.(\lambda\gamma_i + \delta\gamma_i))$;
        $u$-th row of $\Gamma_U$ is set to $\gamma'_u$, $i$-th row of $\Gamma_I$ gets $\gamma'_i$;
    end
end
```
Algorithm 6: Stochastic gradient descent for non-negative matrix factorization of the partially observed ratings matrix.

the learning rate μ is not fixed but decreases[3] over time to help the convergence of the SGD [Bac14]. Also item and user parameters have each their own learning rate μ and penalty weight λ (resp. (μ_i, λ_i) and (μ_u, λ_u)) which are all set using a grid search and the performance on the validation set.

4.1.2 Text summarization for consumer reviews

Early reference work [HL06] on consumer reviews has focused on global summarization of user reviews for each item. The motivation of this work was to extract the sentiments associated to a list of features from all the item review texts. The summarization took the form of a rating or of an appreciation of each feature. Here, contrarily to this line of work, the focus is on personalized item summaries for a target user. Given the difficulty of producing a comprehensive natural synthetic summary, we have considered here extractive summarization which consists in selecting from a set of documents a set if sentences that summarize the documents.

Evaluation of summaries is challenging: how to assess the quality of a summary when the ground truth is subjective? However, this is not a problem to be dealt with in our context. Our goal is to generate a summary that describe the experience of a user concerning an item. As we have access to the actual text written by that user about this item, in our context, this actual text is our ground truth. We have used ROUGE-n [Lin04] (n from 1 to 3) to measure the quality of our generated summaries with respect to the actual review texts. ROUGE-n is a commonly used metric for summarization tasks that computes the ratio of n-grams of the ground truth summary that are found in the candidate summary.

[3]We multiply the learning rate μ by a factor smaller than 1 after each epoch in this series of experiments, here 0.97.

4.1.3 Mixed approaches combining texts and ratings

Considering hybrid models mixing text and ratings is quite rare in recommender systems, principally due to the very recent availability of large datasets [HL04; ML13b] where both modalities are present. A first hybrid model was proposed by [Gan+09]: it is based on hand labeling of review sentences (topic and polarity) to identify relevant characteristics of the items. Topics and polarities are used to learn classification models that are combined to predict the rating. Our approach does not need such hand labeling and as so scales to larger datasets. [ML13a] pushes further the exploitation of texts, by using a joint latent representation for ratings and textual content with the objective of improving the rating accuracy. We propose here various approaches to extract text profiles. Also, the use of a constraint to link the two representation spaces extracted from ratings and texts in [ML13b] makes the optimization procedure cumbersome. We argue here that excellent performance can be achieved using raw representation of text, that scale easily. Also, [ML13b] is focused on rating prediction and do not consider delivering additional information to the user, like our predictive summaries. Very recently, [Zha+14] has considered adding an explanation component to a recommender system. For that, they propose to extract some keywords from the review texts, which are supposed to explain why a user likes or dislikes an item. This is probably the work whose spirit is closest to ours, but the components of their system are only juxtaposed with no common variables of parameters and keyword generation is difficult to evaluate.

[HL04; HL06] combined opinion mining and text summarization on product reviews with the goal of summarizing the qualities and defaults of the items. [Tan+12] proposed a system for delivering personalized answers to user queries on specific products. They built the user profiles relying on topic modeling without any sentiment dimension. [Aga+11] proposed a personalized news recommendation algorithm evaluated on the Yahoo portal using user feedback. This last reference is quite different from our setting and it does not investigate ratings or summarization issues. Overall, we propose here to go beyond a generic summary of item characteristics by generating for each user a personalized summary that is close to what they would have written about the item themselves.

For a long time, sentiment classification has ignored the user dimension and has focused for example on the conception of "*universal*" sentiment classifiers able to deal with a large variety of topics [Bli+07]. Considering the user has been studied only very recently. [Tan+11] for example exploited explicit relations in social graphs for improving opinion classifiers, but their work is only focused on this aspect. [ML13b] proposed to distinguish different rating behaviors and show that modeling the review authors in a scale ranging from connoisseur to expert offers a significant gain for an opinion classification task. Again they focused on sentiment classification only.

4.2 Enriching collaborative filtering with texts profiles

Let us begin by presenting our first contribution: integrating texts along ratings in collaborative filtering. As explained above, our data points are reviews: quadruplets composed of a rating r_{ui}, a user u, an item i and a text document $\mathbf{d}_{ui}$[4]. Our goal is to predict unobserved ratings from observed reviews and we proceed by learning the parameters Θ of prediction function $g_\Theta(u,i)$. We define this function as a generic combination of 4 models:

$$g_\Theta(u,i) = \lambda_0 g_0 + \lambda_1 g_1(u) + \lambda_2 g_2(i) + \lambda_3 g_3(u,i) + \lambda_4 g_4(u,i) \qquad (4.9)$$

The first three terms are biases, in the following order: g_0 the overall bias, g_1 the user bias and g_2 the item bias. The fourth term g_3 is the matrix factorization of the partially observed rating matrix. We have presented these models in the previous section. The fifth term g_4 is the heart of our contribution and takes into account text reviews (documents doc). Lambda parameters $(\lambda_i)_{0\leq i\leq 4}$ control the contribution of each model in the final prediction. For the fifth term, we present three different expressions, thus three different models. The first one is based on bag of words representations of user and item texts. The second one uses latent representations instead of the bag of words representations. The third one uses three bag of words per user and per item: one for all texts, one for positive texts and one for negative texts. A natural fourth model would be to use three latent representations per user and per item: one for all texts, one for positive texts and one for negative texts. The performance of the first and second models are similar and so are the performance of the third and fourth models. Thus we report only the performance of the first, second and third models to focus on the gain of considering multiple representation per user and item, based on the polarity of the texts.

4.2.1 Building text profiles

Current collaborative filtering approaches commonly leave out the texts $\mathbf{d}_{ui}$ of user reviews and only [Gan+09] and [ML13a] do consider the text, to the best of our knowledge. Only the latter provides a model extracting text profiles along with rating profiles. Both profiles are latent profiles, the former is extracted with Latent Dirichlet Allocation [Ble+03] on the texts and the latter is extracted with a classic matrix factorization. However, [ML13a] impose a link between the two latent spaces. If the approach is very elegant, the training protocol is quite cumbersome

[4]We represent texts using vectors that are indicated by bold letters throughout the thesis.

as alternates two training algorithms. We propose here to simplify it by disconnecting both representations. We also argue that a systematic projection is not often the best approach: performance are similar to raw bag of words models but at a greater computational cost.

Raw text approach

We propose here a first, simple approach. To create a text profile for user u, we simply concatenate all documents $\{\mathbf{d}_{ui'}, i'\}$ that user u has written and extract a binary bag of words representation[5] of this document that we denote $\mathbf{d}_u$. The same representation is computed for items and denoted $\mathbf{d}_i$ for item i. To evaluate the *match* between user u and item i, we simply use the cosine:

$$g_{4,T}(u,i) = \cos(\mathbf{d}_u, \mathbf{d}_i) = \frac{< \mathbf{d}_u, \mathbf{d}_i >}{\|\mathbf{d}_u\|\|\mathbf{d}_i\|} \tag{4.10}$$

Combining this model with the other four terms coming from collaborative filtering gives us our first model, that we denote g_T:

$$g_T(u,i) = \lambda_0 g_0 + \lambda_1 g_1(u) + \lambda_2 g_2(i) + \lambda_3 g_3(u,i) + \lambda_4 g_{4,T}(u,i) \tag{4.11}$$

LDA based approach

The second model we propose uses LDA to learn latent representations of texts, and build topic models of comments. Training texts are represented as binary bag of words and a LDA is trained on these bag of words using [PN07]. We use the LDA model as a projector ψ that can take any text (as a binary bag of words) as its input and outputs a vector: the distribution of topics for the input text, as described in [Ble+03]. We then simply feed the binary bag of words $\mathbf{d}_u$ and $\mathbf{d}_i$ to this LDA model to obtain latent representations for the texts of user u an item i. The prediction is done using the dot product, which computes the topic similarity of reviews[6]:

$$g_{4,L}(u,i) =< \psi(\mathbf{d}_u), \psi(\mathbf{d}_i) > \tag{4.12}$$

Combining this models with the other four terms from collaborative filtering, as for g_T, gives us our second model, that we denote g_L:

$$g_L(u,i) = \lambda_0 g_0 + \lambda_1 g_1(u) + \lambda_2 g_2(i) + \lambda_3 g_3(u,i) + \lambda_4 g_{4,L}(u,i) \tag{4.13}$$

[5]This representation focuses on extracting vocabularies without considering frequencies.
[6]This computation is the same as the cosine if ψ profiles are normalized

Sentiment based approach

The third, and last, model that we propose here refines the two previous models by adding the notion of sentiment. Ratings indicates the opinion of users on items. They can be translated into polarities: positive and negative opinions about items. We do so by considering ratings lower than 3 as negative opinions, discarding those equal to 3 as motivated in [PL08], and considering ratings greater than 3 as positive opinions. We then create three profiles for user u:

$\mathbf{d}_u$ is the same as before: binary bag of words extracted from all texts of the user u.

$\mathbf{d}_u^{(+)}$ is the binary bag of words extracted on all positive texts of the user u: that is texts with rating $r_{ui} > 3$.

$\mathbf{d}_u^{(-)}$ is the binary bag of words extracted on all negative texts of the user u: that is texts with rating $r_{ui} < 3$.

In the same way, profiles $\mathbf{d}_i$, $\mathbf{d}_i^{(+)}$ and $\mathbf{d}_i^{(-)}$ are computed for items This model could be derived for both the raw and latent representations of texts. We chose to report the results only for its implementation with raw text representations both models have equivalent performance. The rating prediction is based on all nine matchings of the six vectors, three for the user and three for the item:

$$\begin{aligned} g_{4,S}(u,i) \quad = \quad & \lambda_{s1}\cos(\mathbf{d}_u, \mathbf{d}_i) & +\lambda_{s2}\cos(\mathbf{d}_u, \mathbf{d}_i^{(+)}) & \quad +\lambda_{s3}\cos(\mathbf{d}_u, \mathbf{d}_i^{(-)})+ \\ & \lambda_{s4}\cos(\mathbf{d}_u^{(+)}, \mathbf{d}_i) & +\lambda_{s5}\cos(\mathbf{d}_u^{(+)}, \mathbf{d}_i^{(+)}) & \quad +\lambda_{s6}\cos(\mathbf{d}_u^{(+)}, \mathbf{d}_i^{(-)})+ \\ & \lambda_{s7}\cos(\mathbf{d}_u^{(-)}, \mathbf{d}_i) & +\lambda_{s8}\cos(\mathbf{d}_u^{(-)}, \mathbf{d}_i^{(+)}) & \quad +\lambda_{s9}\cos(\mathbf{d}_u^{(-)}, \mathbf{d}_i^{(-)})) \end{aligned} \tag{4.14}$$

Combining this models with the other four terms from collaborative filtering, as for g_T and g_L, gives us our third model, that we denote g_S:

$$g_S(u,i) = \lambda_0 g_0 + \lambda_1 g_1(u) + \lambda_2 g_2(i) + \lambda_3 g_3(u,i) + \lambda_4 g_{4,S}(u,i) \tag{4.15}$$

4.2.2 Experiments

We will now continue by presenting the dataset on which a series of experiments is conducted as well as the training protocol. We will then discuss the results.

Datasets

This series of experiments is conducted on datasets that are extracted from real websites: Amazon [JL08] and Ratebeer [McA+12]. From these datasets, we extracted many subsets by filtering out users activity and items popularity: smaller datasets are restricted to the most active users and to the most commented items. For each dataset, we randomly split the reviews into three sets: training, validation and test set. The number of reviews, users and items of all datasets can be found in table 4.1.

Name	#Users	#Items	#Training	#Validation	#Test
RB_U50_I200	52	200	7200	900	906
RB_U500_I2k	520	2000	388200	48525	48533
RB_U5k_I20k	5200	20000	1887608	235951	235960
RB_U30k_I110k	29265	110364	2339296	292412	292415
A_U200_I120	213	122	984	123	130
A_U2k_i1k	2135	1225	31528	3941	3946
A_U20k_I12k	21353	12253	334256	41782	41791
A_U210k_I120k	213536	122538	1580576	197572	197574
A_U2M_I1M	2135360	1225387	4642808	580351	580357

Tab. 4.1.: Description of the datasets in terms of number of users, items and reviews in training, validation and test set. RB stands for RateBeer and A for Amazon. The name of a dataset is composed as these initials to indicate the provenance of the data, followed by the number of users and the number of items. For instance, RB_U50_I200 is the dataset extracted from RateBeer containing the top 500 users and top 200 items in number of reviews.

Training protocol

We will now present how all models were trained. For the three biases, no training procedure is necessary: they only involve computing averages of ratings. The matrix factorization term is trained using the stochastic gradient algorithm presented in section 4.1.1 (algorithm 6). Our regularization framework is composed of two parts: factors Γ_U and Γ_I are non-negative and a strong $L2$ penalty is imposed on the parameters. Without this regularization, the high variance of the matrix factorization on a partially observed matrix leads to over-fitting. The use of the bias terms also prevents over-fitting. Please note from table 4.1 that the number of users and items are quite different in the datasets. To take this into account, we use different learning rates and penalty weights for users and items parameters. All the parameters of the SGD are fine-tuned using the MSE on the validation set. For the texts, we build a dictionary removing terms appearing in less than 10 documents. For the LDA, we used the excellent implementation of [PN07].

The bias models are still denoted g_0, g_1 and g_2 for the overall, user and item biases respectively. The models g_C, g_L, g_T and g_S are composite models that integrate the biases term, the matrix factorization g_3 and, for the three later, our contributions.

- $g_C(u,i)$ corresponds to $\lambda_0 g_0(u,i) + \lambda_1 g_1(u,i) + \lambda_2 g_2(u,i) + \lambda_3 g_3(u,i)$ where g_3 is the non-negative matrix factorization learned with stochastic gradient.

- $g_T(u,i)$ corresponds to $\lambda_0 g_0(u,i)+\lambda_1 g_1(u,i)+\lambda_2 g_2(u,i)+\lambda_3 g_3(u,i)+\lambda_4 g_{4,T}(u,i)$ where $g_{4,T}$ is the raw text model.

- $g_L(u,i)$ corresponds to $\lambda_0 g_0(u,i)+\lambda_1 g_1(u,i)+\lambda_2 g_2(u,i)+\lambda_3 g_3(u,i)+\lambda_4 g_{4,L}(u,i)$ where $g_{4,L}$ is the LDA model that incorporate texts in the ratings prediction.

- $g_S(u,i)$ corresponds to $\lambda_0 g_0(u,i)+\lambda_1 g_1(u,i)+\lambda_2 g_2(u,i)+\lambda_3 g_3(u,i)+\lambda_4 g_{4,S}(u,i)$ where $g_{4,S}$ is the raw text model taking sentiments into account.

All parameters λ_j are found by minimizing the loss on the validation set. This allows good generalization. It corresponds to solving a linear system: $\mathbf{\Phi}.\lambda = \mathbf{r}$ where $\mathbf{\Phi}$ is the matrix of prediction (one model per column, one review per row), **lambda** is the vector of parameters to fit and $\mathbf{r}$ is the vector of actual ratings of the validation reviews. We use linear algebra solvers to solve this optimization problem.

Recommender system performance

We will now discuss and compare the performance of our models on the rating prediction task. To do so, as in [ML13a], we use the MSE on the m examples of the test set:

$$MSE(g) = \frac{1}{m}\sum_{(u,i)}(g(u,i) - r_{ui})^2 \tag{4.16}$$

Performance of all model can be found in table 4.2. They confirm our thought about bias models. The worst model is the overall bias g_0 that is too simple to provide good predictions. It is worth mentioning that the item bias g_2 performs better than the user bias g_1: it corresponds to the fact that the variance of ratings given by a user is usually greater to the one of the ratings given to an item. As it was experimentally shown many times in the literature, using collaborative filtering improves performances. As expected, g_C, combining all biases and the matrix factorization, is almost always better than biases. The only exception is for the smallest Amazon dataset where the factorization failed to generalize. Finally, using text for the prediction always improves performance.

Both models without sentiment, g_T and g_L have similar performance. This is surprising as g_T is much more efficient in terms of computations. Our last model, g_S performs most often better than g_T and g_L. We believe the gain comes from the integration in g_S of a polarity through the separation of positive and negative reviews that is not present in g_L. Firstly, training the LDA is long (around 10 hours for largest databases) but also the representation is denser. Cosine of the raw text representation exploit the high sparseness of the bag of words and are faster.

Base	g_0	g_1	g_2	g_C	g_T	g_L	g_S
RB_U50_I200	0.6758	0.6533	0.2091	0.1916	**0.1933**	0.1939	0.1951
RB_U500_I2k	0.5685	0.5256	0.2509	0.2204	0.2202	0.2202	**0.2190**
RB_U5k_I20k	0.6774	0.5878	0.3079	0.2315	0.2292	0.2287	**0.2253**
RB_U30k_I110k	0.70296	0.60644	0.34876	0.2479	0.2412	0.2409	**0.2389**
A_U200_I120	1.5348	1.5658	1.4916	1.7636	1.3560	1.3734	**1.3382**
A_U2k_I1k	1.5316	1.3043	1.2785	1.0989	**1.0539**	1.0567	1.0614
A_U20k_I12k	1.4711	1.2858	1.2361	1.1098	1.0498	1.0501	**1.0452**
A_U210k_I120k	1.5072	1.4454	1.3223	1.1872	1.1230	1.1233	**1.1181**
A_U2M_I1M	1.6051	1.6313	1.4928	1.2700	1.2013	1.2017	**1.1961**

Tab. 4.2.: Test mean squared errors of models on datasets. Each row corresponds to a dataset, each column to a model. The models that we propose, using text profiles, are emphasized in blue (1st row) and are the three last columns.

One surprising fact is the evolution of the performance with the size of the dataset. It appears that – roughly – the bigger the dataset is, the worse the models are, which is the opposite of the expected behavior. However, it has a simple explanation: smaller datasets contains most active users and most commented items for which lots of reviews are available to estimate robust profiles. Larger datasets have more noise and contain users and items with but a few reviews for which parameters are harder to learn.

Explaining gains & losses

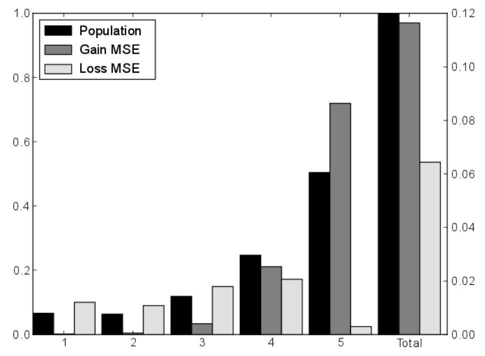
We will now study in detail the predictions to understand why our text model g_S outputs better predictions. We define the gain (4.18) and loss (4.19) in MSE to compare the prediction of the collaborative filtering model g_C and our best text model g_S. A gain corresponds to a better estimation, a loss to a worse estimation, of the text model with respect to the collaborative filtering model. These gain/loss measures are average on all predictions.

$$\delta(u,i) = (g_S(u,i) - r_{ui})^2 - (g_C(u,i) - r_{ui})^2 \tag{4.17}$$

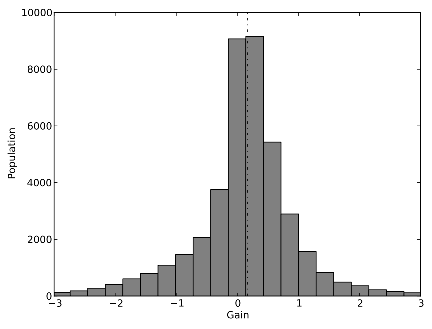
$$Gain_{\text{MSE}} = \frac{1}{m}\sum_{(u,i)} |\delta(u,i)|_+ \tag{4.18}$$

$$Loss_{\text{MSE}} = \frac{1}{m}\sum_{(u,i)} |\delta(u,i)|_- \tag{4.19}$$

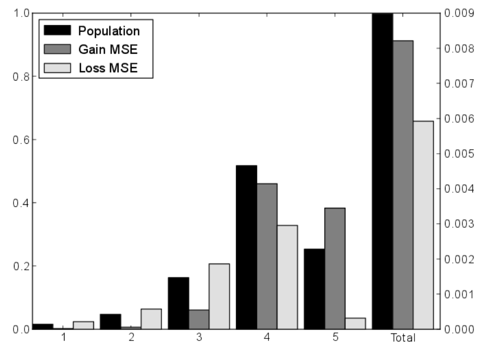
A current bias on recommender system datasets is that most reviews have positive ratings. This mostly comes from the fact that most people mildly satisfied or disappointed do not bother to review. For our datasets, it is also the case. We provide here an in depth analysis of our predictions on two datasets: A_U20k_I12k and RB_U5k_I20k. Figures 4.2a and 4.2c represent the population per rating. We can see that most reviews are above three. The figures also represent the gain and loss per rating of our model g_S with respect to g_C as defined above. The interesting fact is that out text model is better at predicting good ratings and a bit worse at predicting low ratings. However, the bias in the ratings distribution favors our model. Figures 4.2b and 4.2d represent the values of δ. In both cases, the median is positive: for most predictions our model is better.



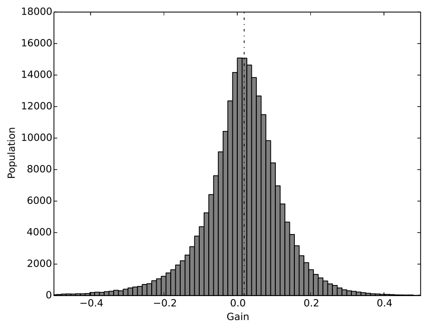
(a) **Amazon** Population and overall gains and losses of MSE per note



(b) **Amazon** Histogram of $\delta(u,i)$ for test reviews and associated median (0.164)



(c) **RateBeer** Population and overall gains and losses of MSE per note



(d) **RateBeer** Histogram of $\delta(u,i)$ for test reviews and associated median (0.019)

Fig. 4.2.: The left column represents populations (black) and MSE gains (gray) and losses (light gray) per rating. The right column represents the distribution of MSE gains/losses, with the median. Overall, our model are better as they estimate the majority ratings 4 & 5 better.

We present in table 4.3 a few hand picked examples that show predictions of g_C and g_S. They exhibit texts containing highly polarized words that benefit to our text model.

Actual rating	5
g_C	4.479377
g_S	4.764128

One of the best historical novels I've read This book is a wonderful tapestry of Norman/Angevin England and Wales. The characters are well-developed and complex. For example, historical treatments of King John invariably cast him as a villain, but here we see him as a character with many facets. The plot follows Joanna, or Joan, the illegitimate daughter of John, through her life from about age five to her late thirties. A reader of this book will learn much about culture clash, women, the Angevins, and England and Wales in the Middle Ages. The book is captivating – I was hardly able to put it down

Actual rating	1
g_C	1.669823
g_S	0.700161

Not taking it back. After comparing the print quality in best mode to my HP 970 CSE inkjet in best mode from the same source there is no comparison. The HP wins in print quality hands down. The CX5200 with its pigment ink is printing unsaturated colors and not sharp in best mode on my first day of use. The HP dye based ink colors are deep and the print is super sharp. I'm not taking this machine back to the dealer for a refund because the wife says the long life durabright ink is required for her scrapbooking. The software install is buggy on an XP home machine and the software is fairly worthless as well. Fortunately my MS Picture It that came with the Dell works with the scanner.

Actual rating	1
g_C	1.296613
g_S	0.900824

Man! This one gave me a hemorrhoid This is just an awful attempt at making music. This guys music literally irritates my [ears] when I hear it. What is really messed up about the whole situation is this guy is polluting the minds of the children with the poor lyrics and ignorant subject matter.

Actual rating	4
g_C	2.963587
g_S	3.619592

Enjoy after repeated Play After spending hours actually forcing myself to listen to this CD, I have to begrudgingly admit that Alicia Keys MAY deserve some of the accolades she has received. The CD is set up so that each song compliments the one before. This is a nice album to mellow out and chill with.

Tab. 4.3.: Test examples of A_U20k_I12k where g_S outperforms g_S. Each cell presents the rating of the review, the text of the review and the predictions of g_C and g_S. These examples stand out as the text strongly impacts the decision of g_S due to the presence of strongly opinionated words.

Actual text Great story and characters; often mannered writing I am going to weigh in very briefly on this book. It has a great story, but it is marred by Arundhati Roy's too frequent displays of mannerism. Many of the characters are very original and the story is full of credible twists and turns, but every thirty pages or so, Ms. Roy lapses into several pages of writing in apretentious stream-of-conscience/surreal style that soon had me skimming toward the next section of narrative substance. Ms. Roy must have felt that she needed to show off how well she could write, but she ended up underscoring the fact that this was her first novel. **Selected sentences** The individual stories of at least twelve characters are told and each story would be rather simple but the stories are all shuffled together with no regard for tense and this makes the book seem much more complex than it actually is. Plus she moves forwards and backwards and sideways in time towards a central event which has been hinted at in countless ways but by the time you get to that event you are mad because all of the confusion could so easily have been avoided if she'd simply told the story, or each of the twelve stories, chronologically. At times the repetitions and sentence fragments and other affectations become more of a hindrance than a benefit, but it seems that some Indian writers feel compelled to write in this sort of native style, and if it is inevitable, then better Roy's fairly controlled method than Rushdie's incomprehensible over-the-top method.	**Actual text** This is a good read, it is a beautiful epic whose true force lies in the poignant details of its characters, richly detailed, woven into a wonderful tapestry. You all so get three books in one, a good bargin, and a good story for teen readers. OVERALL SCORE: (B+) READABILITY: (B), PLOT: (B-), CHARATERS: (A-), DIALOGUE: (B-), SETTING: (B+), ACTION/COMBAT: (B-), MONSTERS/ANTAGONISTS: (B-), ROMANCE: (B+), SEX: (n/a), AGE LEVEL: (PG) **Selected sentences** Terry Brooks is probably the most talented of the derivative-of-Tolkien authors, a pleasant read for those who enjoy basic fantasy tales with a few original bits among the "Tough Guide to Fantasyland" plotlines. Best of the subgenre Terry Brooks' Shannara series was one of the first doorstopper series, varying in originality and in quality but good brain-candy fun. Now the first three books of his trilogy are rereleased in a hardvcover three-in-one edition, not recommended for people with weak wrists, but for those who like good covers and big heavy tomes.
Actual text Has a pitch black pour with a super thick brown bubbly foam head. The head retention is just rediculous, sticks around for a long time. The aromas I got were chocolate malts, coffee, and a little bit of honey. The taste has a medium body mouthfeel to it, with a bitterish finish to it. From the first whiff you know exactly what its going to taste like. Taste like heaven. **Selected sentences** The dark fruits that tend to dominate early on are still detectable (almost as if this were some weirdly lagered impy, shading into a Baltic porter) but are integrated superbly with the roasted malts, giving this one a really unique quality. bottle, thanks to SS, black walnut color with soapy tan foam, aroma has a lot of alcohol and coffee, flavor is well balanced smoked meat, coffee, dried plum, cocoa, finish is well balanced with malty cocoa and coffee The taste is just what I expected from DR; a amazingly smooth silky body, tons of dark fruits, brown sugar, some dry roasted malts, coffee, and a bitter dark chocolate finish	**Actual text** Pours a hazy reddish brown with a nice tan head. Aroma of yeast, fruits, caramel, spices, malt. Flavor is fruit, caramel, yeasts, rasins and malt. Good stuff. **Selected sentences** The taste is medium sweet, with a Belgian yeast, roasted caramel malts, and some fruity notes The nose is fairly sweet, with caramel malts, some dark fruits, Belgian yeast, and spice Appearance: The pour is a cloudy dark orange-brown with a thick and creamy, lasting, lacy beige crown Smell is one complex mother of a dubble, quite unique, good smokeyness comes through with bacon and smoked dried oranges, sweet smokey malts, other dark but dullish fruit of plum and raisin, candied toffee notes and what I'm guessing is alot of wild yeast and a bit of cherry as well.

Tab. 4.4.: 4 hand-picked examples of comments written by users, reported under **Actual text** in each cell, and sentences written by other users on the same item selected using a cosine similarity between the text profile of the user that has written the actual text and the bag of words representation of the sentence. These examples motivated us to provide a proper framework for a task consisting of predicting how a user will comment an item.

Selecting relevant sentences

One interesting by-product of our hybrid models is the possibility of extracting relevant sentences with respect to a review of user u on item i. The simple idea is to use the text profile of u to select sentence written by other users on item i and compare them to the actual text review $\mathbf{d}_{ui}$. The selection is done looking at the cosine between $\mathbf{d}_u$ and the binary bag of words representing sentences of text reviews $\mathbf{d}_{u'i}$. A sample of such hand-picked sentences, without any proper formalism, is presented table 4.4.

4.3 Extending recommender systems

This section describes our second contribution: a new task for recommender system, with an appropriate evaluation framework. This task is the prediction of the text $\mathbf{d}_{ui}$ written by user u on item i. Generating natural text is out of reach, so we will use extraction techniques from automated summarization [AU07]. We will use recommender systems to select among sentences written by other users on item i these which are relevant to user u. We will provide different scoring functions to reflect this relevance as well as different aggregation procedures to generated the final text with the relevant sentences. The point of the study is to generate a synthetic document $\hat{\mathbf{d}}_{ui}$ that is close to the actual text $\mathbf{d}_{ui}$. We quantify the quality of our generated documents using the ROUGE measure [Lin04] used to assess the qualities of summaries [AU07].

The purpose of this task is not the generation of a synthetic document in itself but rather to provide a framework to assess the capabilities of recommender to select from comments written by other users these who reflect the most the style and preferences of a user. Selected comments can be, for instance, presented to the user as part of the recommendation process. The hypothesis behind it is that text conveys more information than ratings, in particular, about the opinion and preferences of users. And we believe that this information should be exploited by recommender systems.

4.3.1 Definition of the models

We first define the recommender systems that we will use for this work. Compared to these presented in the previous section, we modified how we enrich the profiles with text. In the previous sections, our hybrid systems could be considered as hybridizations of collaborative filtering and content-based methods, the descriptors required for the content-based part being extracted from the comments. Here, we will propose models that aim at detecting communities of users. We assume that users of a community have similar writing styles and tastes and that we can exploit these similarities both for the rating prediction and the generation of a synthetic document that we have defined earlier.

The overall bias g_0, the user bias g_1, the item bias g_2 and the matrix factorization g_3 are the same as for the previous series. We propose to integrate texts in the following way:

$$g_4(u,i) = \frac{1}{m_i} \sum_{(u',i)} r_{u'i} \sigma_t(\pi'_u, \pi_u) \tag{4.20}$$

As we said, this approach is different than the one in section 4.2 as we consider only text profiles of users and not of items anymore. We aim at detecting communities of users using the hypothesis that they write similarly (same vocabulary, same topics, . . .). We know that the item bias model performs well as, overall, users of a website rate items similarly. This is a refinement of this fact: users of a community rate items similarly.

In equation (4.20), we denote π_u the text profile of user u. The function σ_t is a similarity measure on text profiles π'_u and π_u of user u and u': it takes high values for similar users. We use light notations for the sum: the meaning of $\sum_{(u',i)}$ is summing on all reviews made on item i by other users (u') in the training set. The number of such reviews is m_i. We will compare two approaches, one with raw bag of words and the other using representation learning. Each will have different representations of text, of course, but also different similarity measures.

Raw bag of words

The first model that we consider is using raw bag of words to represent the text of each review. A preprocessing step removes all words appearing in less than 10 documents. Then, the $100\,000$ most frequent words are kept. Although the number of features is large, the representation is sparse and scales well. π_u is simply the binary bag of words of all texts of user u. In this high dimensional space, the proximity in style between two users is well described by a cosine function, a high value indicates similar usage of words:

$$\sigma_t(\pi_{u'}, \pi_u) = \pi_{u'}\pi_u / (\|\pi_{u'}\| \|\pi_u\|) \tag{4.21}$$

Latent representation using an auto-encoder

For the latent representation, we trained an auto-encoder, similarly to the approach of [Glo+11] and as a replacement of the LDA used before. We have presented in details auto-encoders in chapter 2. It has two components: a coding operator and a decoding operator denoted respectively cod and dec. The two vectorial operators are learned so as to enable the reconstruction of the original text after a projection in the latent space. Namely, given a sentence $\mathbf{s}_{uik}$, represented as a binary bag of words vector, we obtain a latent profile $\pi_{uik} = \mathrm{cod}(\mathbf{s}_{uik})$ and then, we reconstruct an approximation of the sentence using $\hat{\mathbf{s}}_{uik} = \mathrm{dec}(\pi_{uik})$.

The auto-encoder is optimized so as to minimize the reconstruction error on the sentences $\mathbf{s}_{uik}$[7] of the training set:

$$\text{cod}^*, \text{dec}^* = \underset{\text{cod,dec}}{\text{argmin}} \sum_{\mathbf{s}_{uik}} \|\mathbf{s}_{uik} - \text{dec}(\text{cod}(\mathbf{s}_{uik}))\|^2 \tag{4.22}$$

We use the settings proposed in [Glo+11]: our dictionary is obtained after stop-words removal and selecting the most frequent 5000 words. We did not use a larger dictionary such as the one used for the bag of word representation since it does not lead to improved performance and simply increases the computational load. All the sentences are represented as binary bag of words using this dictionary. The coding dimension has been set to 1000 after a few evaluation trials. Note that the precise value of this latent space is not important and the performance is similar on a large range of dimension values. Both cod and dec use sigmoid units:

$$\begin{array}{lcl} \text{cod}(\mathbf{s}_{uik}) & = & \pi_{uik} = \text{sig}(\mathbf{W}\mathbf{s}_{uik} + \mathbf{b}) \\ \text{dec}(\pi_{uik}) & = & \text{sig}(\mathbf{W}^t \pi_{uik} + \mathbf{b}') \\ \text{sig}(\mathbf{x}) & = & \frac{1}{1+\exp(-\mathbf{x})} \end{array} \tag{4.23}$$

Here, π_{uik} is a vector, $\mathbf{W}$ is a 5000x1000 weight matrix and $\text{sig}()$ is an element-wise sigmoid operator on the vector $\mathbf{W}\mathbf{s}_{uik} + \mathbf{b}$.

As motivated in [ML13a; Gan+09], such a latent representation helps exploiting term co-occurrences and thus introduces some semantic. It provides a robust text representation. The hidden activity of this neural network produces a continuous representation for each sentence accounting for the presence or absence of groups of words.

π_u is obtained by coding the vector corresponding to all the texts written by user u in the past. It lies in a latent word space where a low Euclidean distance between users means a similar usage of words. Thus, for the similarity σ_t, we use an inverse Euclidean distance in the latent space:

$$\sigma_t(\pi_{u'}, \pi_u) = 1/(\alpha + \|\pi_{u'} - \pi_u\|^2) \tag{4.24}$$

4.3.2 Generating a synthetic review

As we explained above, our task requires the generation of a synthetic document. This generation a two step process that we will now present. First, for a given review $(u, i, r_{ui}, \mathbf{d}_{ui})$, we rank sentences from comments $\mathbf{d}_{u'i}$ written by other users

[7]This is the k-th sentence of the review of user u on item i.

$u' \neq u$ on the item i. This ranking is done by using scoring function that output a score for each sentence such that a higher value means a more relevant sentence. Second, once the sentences are ranked, we combine the most relevant ones to create the synthetic document. We will define three different procedures for this combination.

Ranking sentences

We start by presenting how to rank sentences using a scoring function. We consider here, as our target, a test review $(u, i, r_{ui}, \mathbf{d}_{ui})$ for which we aim at estimating both r_{ui} and $\mathbf{d}_{ui}$. We will present how to use recommender systems to rank sentences. The first idea is to use the rating prediction of a recommender system $\hat{r}_{ui}$ to select sentences from other reviews that have similar ratings $r_{u'i} \approx \hat{r}_{ui}$. The more similar the ratings are, the more relevant the sentences are. This is our first scoring function. The second idea is to additionally use the similarity between sentence of other reviews and the text profile of user u. This similarity directly gives a pertinence score: the higher the similarity is, the more relevant the sentence is. We have two means of considering the text in the models that we defined above, these two means can be exploited for this ranking purpose.

Assembling sentences

Now that sentences are ranked according to their relevance, we need a procedure to assemble them. This procedure comes from automated summarization [AU07]: we select the most relevant sentences that are also the most distinct from each others. The distinction is made in the sense of the cosine similarity measure on a bag of word representation of the sentence. This forces the selection of sentences using different words and helps generating synthetic documents that looks more natural and less redundant. A pseudo-code implementation of this selection procedure is given by algorithm 7.

Performance and baselines

To evaluate the quality of the generated synthetic documents, we use the ROUGE measure, introduced in [Lin04]. The ROUGE-n measure counts the ratio of n-grams[8] of the actual text $\mathbf{d}_{ui}$ that are present in the generated synthetic text $\hat{\mathbf{d}}_{ui}$. It is a recall oriented measure and suffers from classic biases of these measures: the

[8] n-grams are a successions of n words, more details are given on n-gram in section 5.1.2

```
**Data**: $u$, $i$, $S = \{(s_{u'i}, r_{u'i}, u')\}, \sigma$
**Result**: $\hat{d}_{ui}$
$s^*_{u'i} \leftarrow \underset{s_{u'i} \in S}{\text{argmax}}(\sigma(s_{u'i}, r_{u'i}, u', u, i))$;
$\hat{d}_{ui} \leftarrow s^*_{u'i}$;
Remove $s^*_{u'i}$ from $S$;
**while** $\text{length}\,\hat{d}_{ui} < \text{averagelength}(u)$ **do**
    $s^*_{u'i} \leftarrow \underset{s_{u'i} \in S}{\text{argmax}}(\sigma(s_{u'i}, r_{u'i}, u', u, i) - \cos(s_{u'i}, \hat{d}_{ui}))$ ;
    $\hat{d}_{ui} \leftarrow [\hat{d}_{ui}, s^*_{u'i}]$;
    Remove $s^*_{u'i}$ from $S$;
**end**
```
Algorithm 7: XS greedy procedure: selection of successive sentences to maximize both relevance, according to the scoring function σ, and diversity. $\hat{d}_{ui}$ is the text generated sentence after sentence.

longer the prediction is, the higher the chances are to catch many n-grams. We will counter this flaw by limiting the size of our prediction in the greedy procedure. The value of n influences what ROUGE actually measure, by changing the length of the subsequences to match. ROUGE-1 focuses on correctly matching themes or topics present in the document and is not far from a cosine on bag of words. ROUGE-2 and ROUGE-3 are more representative of the style. However, the number of 2 and 3-grams available for the prediction is huge and, as a result, values of ROUGE-2 and -3 usually are low. In the following, we will use ROUGE-1, ROUGE-2 and ROUGE-3 as our evaluation measures.

Our contribution consists in using recommender systems to select relevant sentence among the comments written by other user on an item. To the best of our knowledge, we are the first to tackle such a task. Thus, we consider additional baselines to compare our models to. First, to evaluate whether our procedure assembling sentences is pertinent, we propose two other procedures. The former selects only the most relevant sentence, in the sense of the scoring function. The latter uses the scoring function on document rather than on sentences to select the most relevant document. Second, to evaluate the quality of our scoring functions, we compare them to two other means of selecting sentences: a random baseline and an oracle. The former simply randomly select sentences among these of comments written on the same item instead of using a relevance score[9]. The latter selects sentences that have the highest ROUGE-n measure w.r.t to the (unobserved) document $\mathbf{d}_{ui}$. As this model uses the (unobserved) text of the test review from which the prediction is computed, it is an oracle.

[9]This baseline is not as naive as it seems: recommender systems show that most users agree on many product (item bias model).

As a result we have a total of 21 models, mixing the 3 procedures to generate the text and the 7 ways to select texts: 1 random baseline, 3 ROUGE-1/2/3 oracles, 3 recommender system scorings (ratings only, ratings + raw text profiles, ratings + latent text profiles).

4.3.3 Experiments

We will first present the results of the models for rating prediction, as in 4.2. We will then discuss the results of text generators using the 21 models we have described. This series of experiments is conducted on the same dataset as before (table 4.1).

Rating prediction

For the rating prediction, we use the mean squared error, once again. Performance of our models are presented in table 4.5. The analysis for models g_0, g_1, g_2 and g_3 is similar to the one of the previous series of experiments. Once again, using texts improves the performance of recommender systems. Both texts models are able to capture the similarities between users (whether using latent representation or not).

Subsets	g_0	g_1	g_2	g_3	g_{L4}	g_{T4}
RB_U50_I200	0.7476	0.7291	0.3096	0.2832	**0.2772**	**0.2773**
RB_U500_I2k	0.6536	0.6074	0.3359	0.3168	**0.3051**	**0.3051**
RB_U5k_I20k	0.7559	0.6640	0.3912	0.3555	**0.3451**	**0.3451**
A_U200_I120	1.5348	2.0523	1.6563	1.7081	**1.4665**	1.4745
A_U2k_I1k	1.5316	1.4391	1.3116	1.0927	**1.0483**	**1.0485**
A_U20k_I12k	1.4711	1.4241	1.2849	1.0797	**1.0426**	**1.0426**
A_U210k_I120k	1.5072	2.1154	1.5318	1.2915	**1.1671**	**1.1678**

Tab. 4.5.: Test performance (mean squared error) for recommendation. g_0, g_1, g_2 are the overall bias, user bias and item bias baselines. g_3 is the matrix factorization baseline. g_{L4} and g_{T4} are our hybrid recommender systems relying respectively on latent and raw text representations.

Text generation

Our main contribution here however is the framework to evaluate text generation. Table 4.6 reports the results on datasets extracted from RateBeer and table 4.7 reports these on datasets extracted from Amazon. We also present here an aggregated version using histograms in figure 4.3. An histogram corresponds to a text selection entity (whole review text, best single sentence, greedy sentence selection. Groups in the histograms (respectively row block of the tables) are composed of three cells corresponding respectively to the ROUGE-1, -2, -3 metrics. Not surprisingly, the results for the single sentence selection procedure (1S) are always worse than for the other two (CT: complete review and XS: multiple sentences). This is simply because a sentence contains fewer words than a full review and it can hardly share more n-grams with the reference text than a longer text. For the Ratebeer datasets, our greedy procedure assembling a set of relevant sentences clearly offers a better performance than selecting a whole review in all cases. Texts written to describe beers also describe the tasting experience. Was it in a bar or at home ? Was it a bottle or on tap ? Texts of the community share the same structure and the same vocabulary to describe both the tasting and the flavors of the beer. Most users write short and precise sentences. This is an appropriate context for our sentence scoring model, where the habits of users are caught by our recommender systems.

The performance slightly decreases when the size of the dataset is increased as seen on tables 4.6 and 4.7 in appendix **??**. As for the series of experiments in section 4.2.2, this is in accordance with the selection procedure of these datasets which focuses first on the most active users and commented items. For Amazon, the conclusion is not so clear and depending on the conditions, either whole reviews or selected sentences get the best score. It is linked to the higher variety in the community of users on the website: well structured sentences like those present in RateBeer are here mixed here with different levels of English and troll reviews.

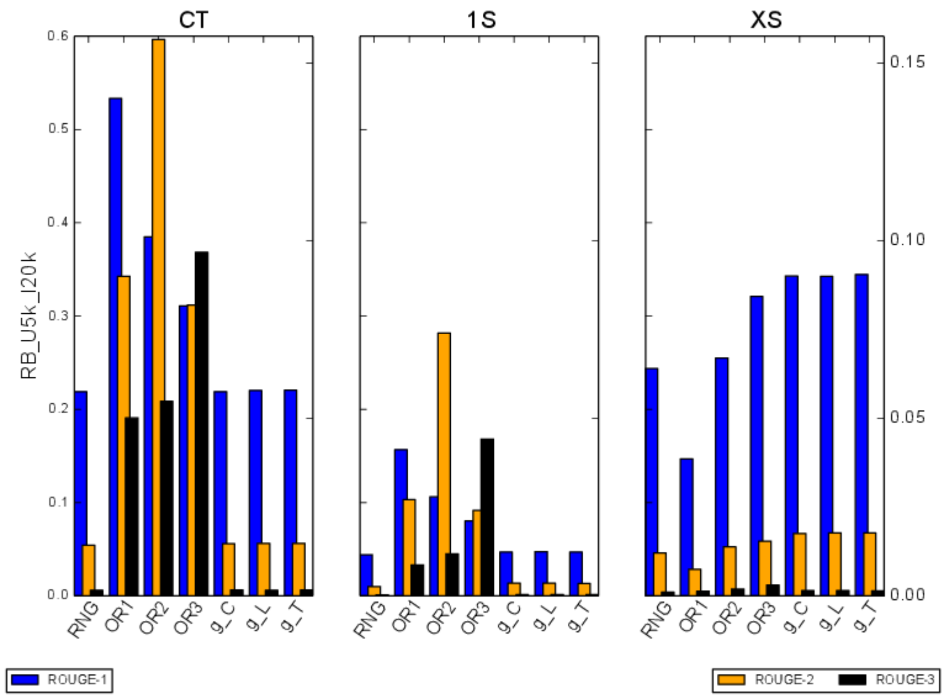
The different models, overall, are following a clear hierarchy. First, stating the obvious, the random model has the worst performance. Then, using a recommender system to select relevant sentences helps in terms of ROUGE-n performance. Overall our models only offer small improvements here with respect to random or NMF text selection. After analyzing this behavior, we believe that this is due to the shortness of the text reviews, to their relatively standardized form (arguments are very similar from one review to another), to the peaked vocabulary distribution of the reviews, and to the nature of ROUGE. This also shows that there is room for improvement on this aspect, for instance by considering additional metrics.

Concerning the oracle several conclusions can be drawn. They are oracle as they use the actual text (that should be hidden) of a review when generating our extractive summary. For both single sentence and complete text selection, the performance gap between oracles and our models is important suggesting that there is still room for improvements here too. For the greedy sentence selection, the gap between the oracles and the hybrid recommender systems is moderate suggesting that the procedure is here fully efficient. However this conclusion should be moderated. It can be observed that whereas, our oracles are effectively an upper bound for single sentence or whole review selection, this is no more the case for multiple sentences selection. Because of the complexity of selecting the best subset of sentences according to a loss criterion (which amounts at a combinatorial selection problem) we have been using a sub-optimal forward selection procedure: we first select the best ROUGE sentence, then the second best, etc. In this case the ROUGE procedure is no more optimal. Concerning the measures, the performance decreases rapidly when we move from ROUGE-1 to ROUGE-2 or ROUGE-3. Given the problem formulation and the context of short product reviews, ROUGE-2 and ROUGE-3 are clearly too constraining.

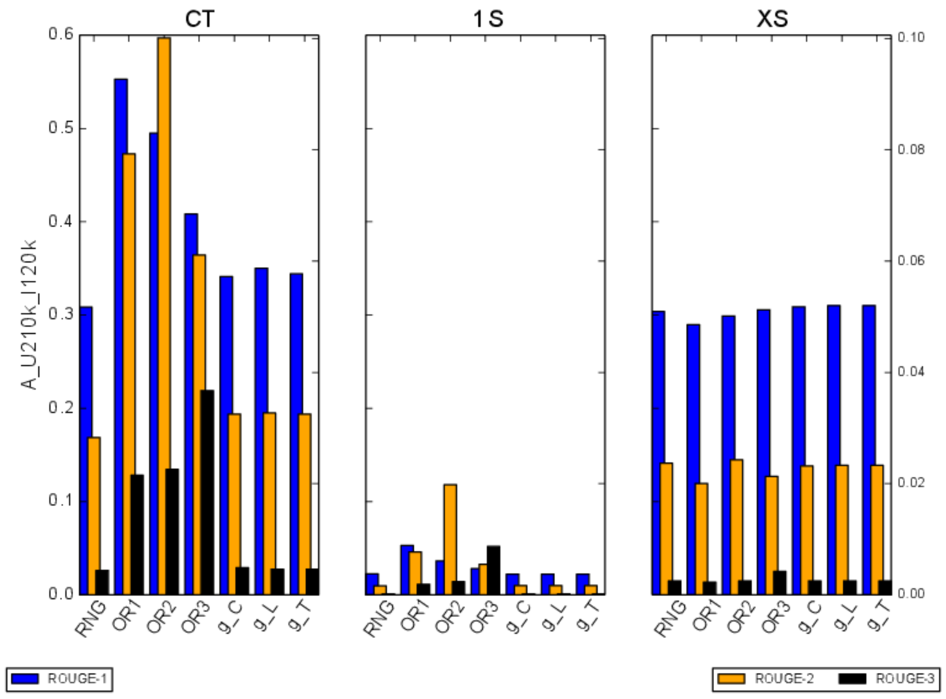
4.4 Conclusion

Recommender systems are a typical application for representation learning and user-centered studies. In the context of rating prediction, that is the most studied recommendation problem in academic research due to data availability, as we said at the begin of this chapter, state-of-the-art collaborative filtering models commonly consider the ratings only. We demonstrated twice the interest of also considering the rich information provided by the text of user reviews. We proposed two formulations. The former has an information retrieval heritage and creates representation of both users and items from texts. The second is deeply user-centered and considers only text profiles for users. Both lead to gains in performance for the rating prediction task. We also proposed a new evaluation framework, to use along with the rating prediction: generating a synthetic text similar to the review that the user would write. To the best of our knowledge, we are the first to propose such an extension. Overall, in our experiments there appears to be little difference between the latent and raw representation in this case. We believe that this is due to the complex nature of text. Recent breakthroughs in natural language processing with representation learning require extremely large text corpora and maybe the datasets we tackled here are big not enough. This work lead to one publication in a conference [Pou+14b] (best paper) and one arxiv publication [Pou+14a]

These studies leave many open questions. Firstly, one natural extension is to provide an unified framework for the two tasks (rating prediction and text generation). Secondly, the ROUGE evaluation can be completed: it is a recall-oriented measure that can be balanced with other ones such as BLEU [Pap+02]. Finally, we did not consider the evolution of user profiles over time as in [ML13b]; it could however improve our text generation performance by including a time dependent similarity so as to take time trends into account.



(a)RateBeer experiments



(b)Amazon experiments

Fig. 4.3.: Histograms of the performances of the summarizer on the two biggest datasets. The scores of the ROUGE-1 metrics are represented in blue while the scores of the ROUGE-2 and ROUGE-3 metrics are in yellow and black. 7 models are compared: the random baseline RNG, the 3 oracles (OR-1, OR-2 and OR-3), the g_C for the collaborative filtering model and the g_L and g_T text-based models. 3 frameworks are investigated: CT (review extraction), 1S (One sentence extraction), XS (Multiple sentence extraction).

	Dataset	RB_U50_I200			RB_U500_I2k			RB_U5k_I20k		
	Performance measure	R-1	R-2	R-3	R-1	R-2	R-3	R-1	R-2	R-3
Complete Text	Random review	0.2339	0.0160	0.0018	0.2321	0.0150	0.0014	0.2190	0.0143	0.0016
	Best ROUGE-1 review	0.4903	0.0843	0.0489	0.5463	0.0976	0.0571	0.5334	0.0900	0.0501
	Best ROUGE-2 review	0.3693	0.1307	0.0512	0.3995	0.1614	0.0614	0.3849	0.1567	0.0548
	Best ROUGE-3 review	0.3106	0.0730	0.0722	0.3263	0.0892	0.1022	0.3108	0.0819	0.0968
	$\gamma_u.\gamma_i$ (best review)	0.2499	0.0178	0.0013	0.2317	0.0159	0.0015	0.2191	0.0147	0.0017
	f_A (best review)	0.2543	0.0183	0.0013	0.2334	0.0160	0.0015	0.2204	0.0148	0.0016
	f_T (best review)	0.2543	0.0182	0.0013	0.2334	0.0160	0.0015	0.2206	0.0148	0.0017
Single sentence	Random sentence	0.0524	0.0026	0.0002	0.0429	0.0026	0.0002	0.0440	0.0026	0.0003
	Best ROUGE-1 sentence	0.1486	0.0221	0.0071	0.1490	0.0256	0.0079	0.1569	0.0271	0.0087
	Best ROUGE-2 sentence	0.0971	0.0587	0.0080	0.1012	0.0704	0.0102	0.1061	0.0740	0.0118
	Best ROUGE-3 sentence	0.0724	0.0151	0.0215	0.0780	0.0228	0.0429	0.0805	0.0241	0.0441
	$\gamma_u.\gamma_i$ (best sentence)	0.0557	0.0045	0.0001	0.0455	0.0034	0.0003	0.0471	0.0036	0.0004
	f_A (best sentence)	0.0556	0.0043	0.0002	0.0456	0.0034	0.0003	0.0472	0.0036	0.0004
	f_T (best sentence)	0.0557	0.0043	0.0002	0.0456	0.0034	0.0003	0.0471	0.0035	0.0004
Set of sentences	Random greedy sel.	0.2785	0.0163	0.0005	0.2508	0.0115	0.0008	0.2437	0.0121	0.0011
	ROUGE-1 greedy sel.	0.5088	0.0576	0.0092	0.5088	0.0576	0.0092	0.1470	0.0075	0.0013
	ROUGE-2 greedy sel.	0.3126	0.0632	0.0062	0.3126	0.0632	0.0062	0.2549	0.0138	0.0019
	ROUGE-3 greedy sel.	0.3972	0.0299	0.0150	0.3972	0.0299	0.0150	0.3210	0.0154	0.0031
	$\gamma_u.\gamma_i$ (sentence, greedy)	0.4251	0.0313	0.0053	0.3450	0.0171	0.0012	0.3428	0.0176	0.0015
	f_A (sentence, greedy)	0.4248	0.0316	0.0041	0.3484	0.0173	0.0012	0.3426	0.0177	0.0015
	f_T (sentence, greedy)	0.4247	0.0314	0.0041	0.3482	0.0173	0.0012	0.3446	0.0177	0.0014

Tab. 4.6.: ROUGE-n evaluation on RateBeer subsets. Top columns are different datasets (see text); R-n is ROUGE-n measure. Row blocks represent generating procedures (CT, 1S, XS). Each row corresponds to a text prediction model.

	Dataset	A_U200_I100			A_U2k_I1k			A_U20k_I10k			A_U200k_I100k		
	Performance measure	R-1	R-2	R-3	R-1	R-2	R-3	R-1	R-2	R-3	R-1	R-2	R-3
Complete Text	Random review	0.3786	0.0393	0.0038	0.3350	0.0337	0.0047	0.3213	0.0304	0.0039	0.3085	0.0283	0.0044
	Best ROUGE-1 review	0.6059	0.0957	0.0245	0.6030	0.0918	0.0204	0.5945	0.0882	0.0205	0.5530	0.0793	0.0215
	Best ROUGE-2 review	0.5642	0.1076	0.0260	0.5511	0.1116	0.0213	0.5371	0.1104	0.0218	0.4950	0.1001	0.0226
	Best ROUGE-3 review	0.4897	0.0815	0.0339	0.4510	0.0691	0.0368	0.4345	0.0661	0.0385	0.4085	0.0611	0.0367
	$\gamma_u . \gamma_i$ (best review)	0.3944	0.0467	0.0041	0.3525	0.0358	0.0050	0.3379	0.0325	0.0042	0.3414	0.0325	0.0049
	f_A (best review)	0.4118	0.0468	0.0046	0.3546	0.0365	0.0051	0.3385	0.0326	0.0042	0.3501	0.0327	0.0046
	f_T (best review)	0.4124	0.0468	0.0045	0.3552	0.0366	0.0051	0.3385	0.0326	0.0042	0.3441	0.0325	0.0046
Single sentence	Random sentence	0.0226	0.0023	0.0006	0.0180	0.0012	0.0001	0.0199	0.0014	0.0001	0.0226	0.0016	0.0002
	Best ROUGE-1 sentence	0.0435	0.0047	0.0007	0.0437	0.0063	0.0016	0.0496	0.0077	0.0020	0.0531	0.0077	0.0019
	Best ROUGE-2 sentence	0.0304	0.0170	0.0018	0.0303	0.0181	0.0024	0.0341	0.0205	0.0027	0.0363	0.0198	0.0024
	Best ROUGE-3 sentence	0.0210	0.0035	0.0041	0.0241	0.0054	0.0086	0.0265	0.0061	0.0100	0.0283	0.0055	0.0087
	$\gamma_u . \gamma_i$ (best sentence)	0.0199	0.0013	0.0004	0.0181	0.0016	0.0001	0.0195	0.0015	0.0002	0.0222	0.0017	0.0002
	f_A (best sentence)	0.0191	0.0016	0.0005	0.0181	0.0016	0.0001	0.0196	0.0015	0.0002	0.0222	0.0017	0.0002
	f_T (best sentence)	0.0191	0.0016	0.0005	0.0182	0.0016	0.0001	0.0196	0.0015	0.0002	0.0222	0.0017	0.0002
Set of sentences	Random greedy sel.	0.3518	0.0325	0.0025	0.3753	0.0323	0.0035	0.3613	0.0298	0.0030	0.3038	0.0237	0.0025
	ROUGE-1 greedy sel.	0.3747	0.0338	0.0022	0.3625	0.0253	0.0024	0.3440	0.0234	0.0024	0.2896	0.0200	0.0023
	ROUGE-2 greedy sel.	0.3615	0.0430	0.0029	0.3650	0.0259	0.0024	0.3544	0.0259	0.0026	0.2988	0.0243	0.0025
	ROUGE-3 greedy sel.	0.3571	0.0341	0.0048	0.3730	0.0272	0.0043	0.3665	0.0260	0.0043	0.3054	0.0213	0.0042
	$\gamma_u . \gamma_i$ (sentence, greedy)	0.3615	0.0329	0.0034	0.3785	0.0309	0.0032	0.3710	0.0296	0.0030	0.3087	0.0232	0.0025
	f_A (sentence, greedy)	0.3589	0.0331	0.0029	0.3794	0.0313	0.0034	0.3726	0.0298	0.0030	0.3103	0.0233	0.0025
	f_T (sentence, greedy)	0.3586	0.0332	0.0029	0.3792	0.0312	0.0033	0.3726	0.0298	0.0030	0.3103	0.0233	0.0025

Tab. 4.7.: ROUGE-n evaluation on Amazon subsets. Top columns are different datasets (see text); R-n is ROUGE-n measure. Row blocks represent generating procedures (CT, 1S, XS). Each row corresponds to a text prediction model.

5 Opinion meets User: Unifying Opinion Classifiers and User Profile Learning

Contents

Recommender systems are, as we said, a typical user-centered models. As for other methods of personalized access to information, they are able to take the preference of users into account, which is a first step towards the consideration of context for machine learning algorithms. We believe that using context can help for many other applications, and in particular for sentiment analysis and polarity classification. In the previous chapter, we have shown the benefit of using the polarity of texts to enrich the profiles extracted by recommender systems, and in this chapter we will consider the opposite: we will show the benefit of using the profiles extracted by recommender systems to contextualize polarity classification models.

The challenge of the polarity classification task is to decide whether texts contain a positive or a negative opinion by assessing the polarity of words. As well described in [PL08], review texts are an important resource for this task: they provide huge quantities of supervised polarized texts, thanks to ratings, that can be used to design generic polarity models. Ideally, those models can be applied to other texts to assess their polarities. We claim that such generic models can benefit from contextual information provided by the analysis of user preferences and writing styles. For instance, the use of irony is very personal, depends on the author's style and can change the polarity of a text.

In this chapter, we will consider the use of a collaborative filtering based recommender system to extract user and item profile and use them to improve polarity classification performances by contextualizing the prediction with respect to the user and the item. We will experiment using classic recommender systems and also the ones introduced in chapter 4. Additionally, we propose a model chaining a recommender system and a polarity classification model. The latter, instead of learning the polarity of words in the vocabulary, is contextualized by the former and extracts a vocabulary highlighting the difference between the expectations of a user and the reality.

5.1 Opinion mining

A complete description of opinion mining and polarity classification can be found in the excellent [PL08]. We will begin this section by presenting the task of polarity classification on user reviews. We will then review various possible representations of the text, from the classic bag of words to recent word tables, that we already used in chapter 4 without a proper definition.

5.1.1 Polarity classification

The long-term goal behind sentiment analysis is to create generic models able to detect sentiments. The most common task, due to data availability is polarity classification. The goal of this task is to predict whether a text contains a positive or a negative opinion. As presented in [PL08], three main classifiers have been used for this task: naive Bayes, logistic regression and linear SVM. We have chosen the latter and will explain how it works later on in the chapter. The domain of opinion classification has benefited from the availability of large datasets of user reviews. These review provide supervised (thanks to the ratings) example of opinionated texts. We will present later on how the text is processed to focus here on how the rating is exploited to create a binary classification problem.

As [PL08] describes, user reviews are typically a text with a rating from 1 to 5. These ratings are converted to binary labels as follows:

- Ratings 1 and 2 are considered as negative examples.
- Ratings 4 and 5 are considered as positive examples.

- Ratings 3 are discarded as it is unclear whether they are positive or negative.

One difficulty arising when considering the task of creating a generic model is the shift in vocabulary among domains. This difficulty is called domain generalization. For instance, words describing good movies are not the same as words describing good refrigerators. The domain of transfer learning studies how to transfer knowledge on one task to another provides techniques to overcome such difficulties. For instance, it is possible to use deep neural network to generalize across domains, using massive unsupervised datasets [Glo+11]. The recent [LM14] extends the word2vec model presented in [Mik+13] to tackle polarity classification [Mik+13] has no representation for documents, only for words, and cannot be used for this task. As presented in chapter 2, [LM14] learns latent representation of documents additionally to latent representation of words. It exploits the semantic proximity of words to overcome the domain generalization problem. However, these approaches all aim at creating a generic model able to classify all texts. The approach that we propose here consists in using the ability of recommender systems to consider context (the preference of users) in the context of polarity classification.

5.1.2 Text representations

We briefly introduce two classic techniques used to represent text as vectors for classification purposes, bag of words and n-grams, as well as associated pre-processing techniques.

Bag of words

The most popular technique for text representation in a classification context is the bag of words. It is used to represent texts of various lengths as vectors of the same length. A vocabulary V (a set of words) is extracted from the texts of the training set, V contains $|V| = n$ words. This vocabulary is often called dictionary, but we will stick to the name of vocabulary to avoid confusion with the dictionary in the context of matrix factorization. Each of the n words defines a dimension, giving a n-dimensional representation $\mathbf{x} \in \mathbb{R}^n$ of each text. In the simplest case, for each text x_i is set to 1 if the i-th word of the vocabulary is present in the texts. Such a representation is called binary bag of words. It is possible to use counts instead of a binary variable for each words. For the term frequency representation, x_i is then set to the number of times the i-th word of the document appears in the text. Many different variants of these weighting schemes have been proposed. Another popular text encoding is *tf-idf* (term frequency and inverse document frequency) where the term frequency is multiplied by a function of the inverse document frequency[1] However, since [PL08], it is known that binary bag of words are well adapted to sentiment classification, which is why we use them in this work.

n-grams

We discussed n-grams through the use of the ROUGE-n measure in chapter 4. This representation scheme is similar to bag of words. However, instead of using single words to build the vocabulary, n-grams are used. A n-gram is a succession of n words from the training set texts. Using subsequent words gives additional information. For polarity information it is possible to catch immediate negations such as *not good* and combinations such as *so bad*. This gain comes from the context that is extracted by using n-grams. However, this gain comes at the cost of an explosion of the dimensionality of the representation. For $|V|$ words, there exist $\binom{|V|}{n}$ possible n-grams. Of course, not all are semantically correct – it is hard to imagine a sentence with *dog computer* if *dog* and *computer* are in the vocabulary – but the number is still great. It is also highly sensitive to spelling mistakes.

Word preprocessing

One important consideration for text representation, when extracting the dictionary (either of words or n-grams) is word preprocessing. One of the first operation is the segmentation of each text as a list of words. For the English language, one strong heuristic is to split on common punctuations (,.-'. . .).

Once text are split in lists of words, one possible step is the removal of stop-words. Words like *a*, *the*, *we*, . . . are part of the language but do not carry information about the topic discussed. This is why they are commonly removed when using topic modeling method like LDA [Ble+03] or auto-encoders [Glo+11]. In the context of polarity classification, however, the usage of such stop-words appears to act as a bias that catches the average polarity of the domain for which the model is trained and this helps achieve better predictions. As usually done for this task, we keep stop words in the vocabulary for our series of experiments.

It is possible to refine further the preprocessing by using stemming and/or lemmatization. Stemming consists in representing each word by its statistical root by implementing basic rules suppressing pre/postfix. For instance *stampeding* can be represented by its root *stamped-*. Lemmatization is closely related. Instead of looking for the root of a word, lemmatization searches for its lemma – its canonical

[1]The document frequency of a word is the number of distinct documents in which the word appears.

form. For instance *worst* will be map to *bad* using lemmatization while being left untouched by stemming. Such preprocessing, as for stop-words removal, are important when looking at the topics discussed in the texts. Using raw input is more effective for polarity classification. As for stop words, the usage of certain words is highly discriminant of the user (language level, spelling errors, . . .) and considering all recurrent versions of each word actually helps models.

5.2 Improving polarity prediction

We will use the different hybrid recommender systems defined in chapter 4 as polarity predictors instead of rating predictors, using simple thresholds on the predicted ratings. We will first use models of section 4.2 to show that improving the quality of user and item profiles improves the performance as polarity predictor as well. Then, using those of section 4.3, we will also compare them to the strong baseline of a linear SVM classifier.

5.2.1 Recommender systems as polarity classifiers

As briefly described above, we restrict ourselves to recommender systems as polarity classification models in this setting. This is an original approach since sentiment classification usually does not use information from the past since it is traditionally a survey tool, not a predictive one.

Classification error rate

The metric that we use in polarity classification is the Classification Error Rate (CER). For a model ϕ and a dataset $\mathcal{D}$ containing m examples, the CER is defined as follows:

$$CER(\phi, \mathcal{D}) = \frac{1}{m} \sum_{(\mathbf{x},y)} \mathbf{1}(y \neq \phi(\mathbf{x})) \tag{5.1}$$

In this formulation, $\mathbf{1}$ is an indicator function that equals one only if the boolean property it takes as input is true and zero otherwise. Basically, the CER counts the ratio of errors made by model ϕ among all examples of dataset $\mathcal{D}$.

Models

We will here give a brief reminder of the models that were introduced in section 4.2. We will use these recommender systems as polarity classifiers in the next series of experiments.

ϕ_0 the overall bias, the average rating on the training set. In the context of polarity classification it always predicts the same class. As we have seen in chapter 4, positive texts are over represented in most user reviews datasets, thus the prediction of ϕ_0 is – for most datasets – always 1.

ϕ_1 the user bias, the average rating per user. Each user will output the same prediction over time, considering here that some users only like items and some only dislike items.

ϕ_2 the item bias, the average rating per item. Here the prediction is the same for an item independently of the user. It considers that there exists good and bad items and that all users respect this separation.

ϕ_C the matrix factorization with all bias terms is the most common model for collaborative filtering, predictions depend both on the item and the user.

ϕ_L our hybrid model including review texts using LDA[2].

ϕ_S our hybrid model including review texts using a raw representation of texts as well as polarity information.

We do not report the performance of ϕ_T as, as for the rating prediction task in the last chapter, they are equivalent to the performance of ϕ_L.

Sentiment classification performance

Classification gives another perspective on the recommender system predictions. Here, predicting 5 (class $+1$) instead of 4 (same class) is not an error. Predicting 4 instead of 2 (class -1) is. The performance of our models as sentiment classifiers can be found in table 5.1. We use the same datasets as in chapter 4, the description of which can be found in table 4.1.

In this context, both the overall bias ϕ_0 and user bias ϕ_1 fail to output correct prediction. The item bias ϕ_2 confirms that it is the best bias model: considering

[2]Once again, the texts used to build the models do not concern the current reviews but only past data are processed here.

Base	ϕ_0	ϕ_1	ϕ_2	ϕ_C	ϕ_L	ϕ_S
RB_U50_I200	18.75	19.11	8.21	8.91	7.62	**7.17**
RB_U500_I2k	18.75	18.60	10.73	10.26	10.16	**9.92**
RB_U5k_I20k	25.03	24.65	14.43	14.32	12.54	**12.42**
RB_U30k_I110k	26.33	25.83	16.05	15.20	13.88	**13.70**
A_U200_I120	17.50	21.67	17.50	25.83	19.23	**16.92**
A_U2k_I1k	15.94	15.16	15.92	14.10	**11.38**	11.68
A_U20k_I12k	14.74	14.47	14.19	14.24	**11.26**	**11.28**
A_U210k_I120k	14.73	15.99	14.81	14.42	12.22	**12.05**
A_U2M_I1M	14.91	16.39	15.78	15.91	13.14	**13.05**

Tab. 5.1.: Classification Error Rates on test datasets.

the average rating of an item is a strong baseline. It correctly reflects the sentiment of most users. Also it is worth mentioning that the matrix factorization ϕ_3 is not significantly better than the item bias. The text models are however better. It is known since [Pan+02] that considering raw text leads to good performances for sentiment classification. So it is not surprising to see the gain coming from considering text. We believe that the gap between the two models is due to the fact that ϕ_S includes a notion of polarity that ϕ_L does not.

5.2.2 Mixing predictions with a linear SVM

The series of experiments from section 5.2.1 confirms the criticality of the richness of profiles extracted by recommender systems to achieve good performance in sentiment classification. We now add a comparison with a strong baseline from the domain and hybrid models mixing recommender systems and polarity classification models. The novelty of this approach consists in mixing models that operate on different time scales: recommender systems consider past information to output the rating of an unobserved review while SVMs (our polarity classification model) observe the current text to predict a polarity.

The series of experiment of confirms that the more informative the profiles are, the better the performance as polarity classifier are, like for the rating predictions. However they lack of a comparison with a strong baseline from the domain, which is what we will do now.

Baseline: linear support vector machine

The first model that we use here, our baseline in terms of performance, is a linear SVM. Because of the high dimensionality and sparseness of text representations, linear classifiers are the most popular learning machines used in most classification tasks, including polarity classification [PL08]. We provide here a short introduction of linear support vector machines in the context of polarity classification. This presentation is of course incomplete as it focus mainly on the cost to optimize, the description of the weight vector and regularization strategies. Complete descriptions can be found in many reviews of the literature like the [Bur98; SS02].

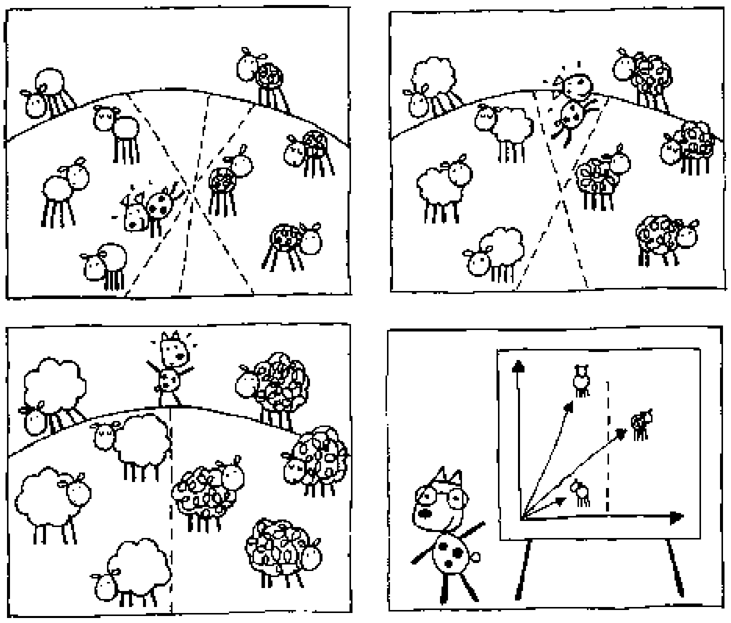


Fig. 5.1.: Illustration from [SS02] of a binary classification problem where data points, the black and white sheep are linearly separable. The sheep dog is looking for an hyperplane separating the two flocks.

The best hyperplane Linear models for classification aim at separating regions of space using an hyperplane. The distance of a set of points to an hyperplane is the minimal value of the distance of the points in this set to the hyperplane. Considering two classes which are linearly separable, the margin is the minimal value of the distance between the hyperplane and each class. The SVM classifier will select the hyperplane that maximizes the margin, thus maximizing the separation between the two classes. This concept is presented, with humor, on the illustration in figure 5.1, extracted from [SS02] where the dog aims at separating the two classes – black and white flocks – using a linear frontier.

As presented in the illustration from [SS02] in figure 5.2, the margin $M(\mathbf{w}, b)$ is:

$$\begin{aligned} \mathbf{x}_1^* &= \operatorname*{argmin}_{(\mathbf{x},y),y=1} \; < \mathbf{x}, \mathbf{w} > -b \\ \mathbf{x}_{-1}^* &= \operatorname*{argmin}_{(\mathbf{x},y),y=-1} \; < \mathbf{x}, \mathbf{w} > -b \\ M(\mathbf{w}, b) &= < \frac{\mathbf{w}}{\|\mathbf{w}\|}, \mathbf{x}_1^* - \mathbf{x}_{-1}^* > = \frac{2}{\|\mathbf{w}\|} \end{aligned} \tag{5.2}$$

We denote by $\mathbf{w}$ the normal vector of the separating hyperplane learned by the SVM and b the bias. The margin is the distance between the points of each class closest

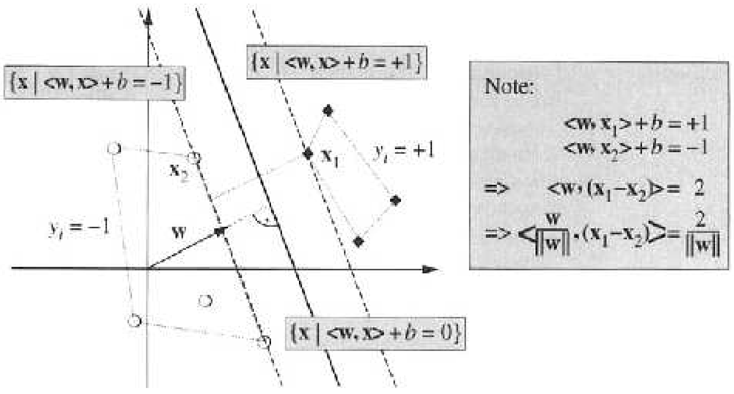


Fig. 5.2.: This plot from [SS02] illustrates the computation of the margin. White points are members of the -1 class and black squares of the $+1$ class. The hard line is the separating hyperplane, dotted lines are parallel hyperplanes passing through the support vectors: points the closest to the separating hyperplane.

to the separating hyperplane and is $\frac{2}{\|\mathbf{w}\|}$ as presented in Maximizing this distance is equivalent to minimizing the norm of $\mathbf{w}$ under the constraint that examples are on the correct side of the hyperplane, respecting the margins. This is formulated as the following optimization problem:

$$\hat{\theta} = \underset{\mathbf{w},b}{\operatorname{argmin}} \ \|\mathbf{w}\|_2^2 \text{ s.t. } \forall(\mathbf{x}, y),\ y(\mathbf{w}.\mathbf{x} + b) \geq 1 \tag{5.3}$$

This formulation is called the primal form of the SVM loss and is used as such in general for linear classifiers. When more complex kernels are used, the dual form is generally preferred. The latter uses Lagrange multipliers for the constraints and Karush-Kuhn-Tucker conditions for the optimization.

Hinge loss In this work, we will use the primal form and for the loss the following surrogate:

$$\hat{\theta} = \underset{\mathbf{w},b}{\operatorname{argmin}} \ \frac{1}{m} \sum_{(\mathbf{x},y)} |1 - y(\mathbf{w}.\mathbf{x} + b)|_+ + \lambda_{\mathbf{w}} \|\mathbf{w}\|_2^2 \tag{5.4}$$

In the formulation of equation (5.4), the constraints $y(\mathbf{w}.\mathbf{x} + b) \geq 1$ of the the primal form in equation (5.3) are directly optimized. The minimization of the norm of $\mathbf{w}$ corresponds the classic $L2$ regularization scheme. This loss formulation can easily be minimized using stochastic gradient descent. It acts as a support vector machine in the sense that only examples that are in the *margin* or ill-classified – such that $y(\mathbf{w}.\mathbf{x} + b) < 1$ – are considered for update. As discussed in [LCB04], using stochastic gradient descent is a good choice for large scale problems, both in terms of computational complexity and generalization of the model. Considering the case of bag of words, even using large dictionaries, the representation is extremely sparse. This leads to efficient computations of the dot product and updates. The $L2$ regularization is necessary to address the high variations in terms frequencies as described in [Raf+12].

Interpretation of the weights In the context of polarity classification, we will consider two possible labels [PL08]: the text is either expressing a positive $(+1)$ or negative (-1) opinion. The prediction of the model, given a vectorial representation $\mathbf{x}$ of a text is:

$$\hat{y} = \text{sign}\,(\mathbf{w}.\mathbf{x} + b) = \text{sign}\,(b + \sum_i w_i.x_i) \tag{5.5}$$

Consider that, as it will be the case for our series of experiment, $\mathbf{x}$ is a binary bag of word. We denote by T the indexes of the words present in the text that $\mathbf{x}$ represent, then:

$$\hat{y} = \text{sign}\,(\mathbf{w}.\mathbf{x} + b) = \text{sign}\,(b + \sum_{i \in T} w_i) \tag{5.6}$$

The last formulation explicits the meaning of weights $\mathbf{w}$. Each weight w_i represented the polarity of the i-th word of the vocabulary and the decision simply is the sum of weights of each word in the text.

$$\hat{\mathbf{w}}, \hat{b} = \underset{\mathbf{w},b}{\text{argmin}}\ \frac{1}{m} \sum_{\mathbf{x},y} |1 - y(\mathbf{w}.\mathbf{x} + b)|_+ + \|\mathbf{w}\|_2^2 \tag{5.7}$$

We use LibLinear [Fan+08] to train the models in our case, selecting the optimal set of parameters based on validation performance – performance on the validation dataset.

Models

For this series of experiments, we will use the models that we introduced in section 4.3. We will briefly describe them here:

ϕ_2 the item bias. It is the only bias model that we consider here as both the overall and user bias models only have poor performance, as seen in table 5.1

ϕ_C considers the matrix factorization as a prediction model.

ϕ_L our hybrid model using a latent representation of texts extracted with an auto-encoder

ϕ_T our hybrid model using a raw representation of texts

The baseline, LibLinear is denoted LL. We will also introduce two additional hybrid models: LL + ϕ_L and LL + ϕ_T. They combine the prediction of the linear classifier and our hybrid models. Predictions are combined linearly *a posteriori*, with combination coefficients that are set on the validation set.

Sentiment classification

We conducted similar experiments as the previous series. But this time we added a strong baseline: a linear SVM – trained using [Fan+08] that is know to give good classification performance [PL08]. The results are reported in table 5.2. From the three biases, we report only the performance of the item bias as both the overall and user biases have low performance. The performance is different from the one in table 5.1 as the splits were different.

Subsets	LL	ϕ_2	ϕ_C	ϕ_L	ϕ_T	LL + ϕ_L	LL + ϕ_T
RB_U50_I200	5.35	5.12	6.01	5.57	5.57	**3.79**	**3.79**
RB_U500_I2k	7.18	10.67	9.73	8.55	8.55	**6.52**	6.92
RB_U5k_I20k	8.44	11.80	10.04	9.17	9.17	**8.33**	**8.35**
A_U200_I120	**10.00**	15.83	22.50	20.00	20.83	**10.00**	**10.00**
A_U2k_I1k	7.89	15.25	12.85	12.62	12.62	**7.5**	**7.5**
A_U20k_I12k	**6.3**	13.99	12.79	12.38	12.37	**6.29**	**6.29**
A_U210k_I120k	**6.25**	14.04	14.40	13.32	13.31	**6.22**	**6.22**

Tab. 5.2.: Test performance (classification error rate) as polarity classifiers. LL stands for LibLinear (SVM), ϕ_2, ϕ_C, ϕ_L, ϕ_T are the recommender systems as in table 4.5. LL + ϕ_L and LL + ϕ_T are two hybrid opinion classification models combining the SVM classifier and ϕ_L and ϕ_T recommender systems.

The hierarchy among recommender systems was expected since the last series of experiments. The main difference is that the linear SVM is always better than the recommender systems at predicting sentiments. This leads to the idea of mixing both predictions. The linear SVM uses the text d_{ui} of the review for which the prediction is done while the recommender systems rely on ratings only. Combining both predictions gives us two additional models LL + ϕ_L and LL + ϕ_T that are performing well on all the datasets.

5.3 Extraction of a surprise vocabulary

Here we modify our experimental setup and present a method to train linear SVMs that compensates mistakes made by recommender systems (and explains these mistakes). Contrary to previous approaches in this chapter, the SVM is not trained as a classifier to predict the polarity of a text but as a regressor to predict the difference between the rating prediction of the recommender system and the actual rating. As a result, its weights will no longer represent the polarity of a word but rather how much this word indicates that the opinion of the user is not the one that was expected, learning a surprise vocabulary.

5.3.1 Model

Our baseline is a linear SVM, as before, trained using the stochastic gradient descent models from the Python library named Scikit Learn (based on Liblinear code). For the recommender system, we use the python API to GraphLab [Low+10].

Recommender and regression machine

We provide here a simple model that combines a recommender system and a linear regression machine. We first train a collaborative system to predict ratings. We use this model to output a rating prediction for all reviews as in equation (5.9). The linear SVM is trained on the difference between the predicted ratings and the actual ratings as in equation (5.10) given the text. That is, given a review $\mathbf{d}, r$ and the rating prediction of the recommender system $\hat{r}$, our SVM must learn a prediction function g such that:

$$g(\mathbf{d}) = r - \hat{r} \tag{5.8}$$

For instance, consider a review $\mathbf{d} =$ "I love this series but was disappointed by this movie" and $r = 3$, where the user like a series of movies, such as Harry Poter, but not this particular movie. As the history indicates that previous episodes were liked by the user, the recommender system will probability predict a rating $\hat{r} \geq 3$ that is greater than the actual rating. We train the SVM so it can learn the words in the text that indicate why this user did not rate the movie as expected. In our example, but and $disappointed$ are such words.

$$\hat{\mathbf{\Gamma}}_U, \hat{\mathbf{\Gamma}}_I = \underset{\mathbf{\Gamma}_U, \mathbf{\Gamma}_I \geq 0}{\text{argmin}} \sum_{(u,i,r_{ui})} (r_{ui} - \gamma_u.\gamma_i)^2 + \lambda_U \|\gamma_u\|_2^2 + \lambda_I \|\gamma_i\|_2^2 \tag{5.9}$$

$$\hat{\mathbf{w}}, \hat{b} = \underset{\mathbf{w}, b}{\text{argmin}} \frac{1}{m} \sum_{\mathbf{x},u,i,r_{ui}} (1 - (r_{ui} - \gamma_u.\gamma_i)(\mathbf{w}.\mathbf{x} + b))^2 + \|\mathbf{w}\|_2^2 \tag{5.10}$$

5.3.2 Results and discussion

We now discuss the results of our series of experiments. We begin by presenting the protocol that we have followed for all datasets and models. We then compare the performance over all dataset of the models used as polarity classifiers with the standard classification error metric. At last, we analyze the weight vectors of our models.

We used a similar protocol for all models across all datasets for which we give the implementation details and the procedures we used to select the parameters for each models later on.

Datasets

Our experiments are conducted on different datasets parsed from real websites. They are all collections of user reviews with both a rating and an associated (rather short) text:

Yelp This dataset is made available for research purposes by Yelp. It contains more than 1M reviews.

RateBeer This dataset was parsed by McAuley and Leskovec [ML13b; ML13a]. It contains 2.9M beer reviews from RateBeer.com.

Movies This dataset was also parsed by McAuley and Leskovec [ML13b; ML13a]. It contains 7.9M movie reviews from Amazon.com.

AmazonLiu This dataset was parsed by Jindal and Liu [JL08], originally for opinion spam detection. It contains 5.8M reviews from Amazon.com.

We focus on the extraction of a surprise vocabulary: words that indicates the user did not rate the item as expected from past ratings. To do so, we focus on users and items for which enough data is available and ignore the problem of cold start. This is the first step of our preprocessing. First, we select users and items having more than 10 reviews. We then split the dataset ensuring that users have similar proportion of reviews in training, validation and test sets. Each user will have 70 % of their reviews in the training set and 15 % for the validation and test sets respectively. The characteristics of the resulting datasets are given in table 5.3.

Name	Users	Items	Training	Validation	Test
Yelp	38665	25384	411735	88229	88229
AmazonLiu	135599	186009	1260457	270098	270098
RateBeer	11732	46935	931226	199549	199549
Movies	258356	129512	4017660	860927	860928

Tab. 5.3.: Description of the datasets used for this series of experiments. Each row corresponds to a different dataset. Columns are, from left to right: the number of users in the dataset, the number of items in the dataset, the number of reviews in the training, validation and test sets respectively.

Text preprocessing

We use a simple procedure for text representation which is known to provide efficient and robust classifiers. We build a vocabulary from the texts of training reviews (reviews in the training set). We limit ourselves to words appearing in more than 10 documents to filter out less frequent spellings or errors in the tokenization. Each text is then presented as a binary bag of words. All words of the vocabulary that match the above criteria are kept. Table 5.4 gives the vocabulary size of each dataset. The column titled *Unique words* gives the number of unique token – here considered as words – in the training reviews. The vocabulary column is the number of words selected in the vocabulary. This is a sparse high dimensional representation.

Name	Training	Vocabulary	Unique words
Yelp	411735	48983	161394
AmazonLiu	1260457	101004	542545
RateBeer	931226	53217	199548
Movies	4017660	269385	447247

Tab. 5.4.: The first column recalls the number of reviews in the training set. The second column gives the number of words in the selected vocabulary, used for the bag of words representation, while the last gives the number of words in the dataset.

Parameter tuning

Our protocol for parameter tuning is the following: we have split the dataset into three sets (training, validation and test sets) and we exploit the validation set to set the hyper-parameters, using grid searches.

For linear classifiers used in the baseline and the combined models, we proceed as follows. We first do a simple grid search, training the model on the training set and computing performance on the validation set. We then select the set of parameters (learning rate, weight of the regularization, number of iterations, . . .) that gives the best validation performance. We finally train the model once more using this set of parameters and examples from both the training and validation sets. The same procedure is done for the non-negative matrix factorization to set the learning rate, rate decay, weight of the regularization, . . .

5.3.3 Polarity classification performances

As for the last series of experiments, we will compare the performance of the models as polarity classifiers. Performances are provided in table 5.5. Three models are presented:

LSVM the linear classification machine on the text, this is our strong baseline.

Surprise our hybrid model that learns a regression machine to correct the outputs of a recommender system.

LSVM + Surprise both models mixed *a posteriori* using a linear combination.

Datasets	LSVM	Surprise	LSVM + Surprise
Yelp	6.07	8.51	5.97
AmazonLiu	6.26	9.51	6.04
RateBeer	8.75	10.22	6.01
Movies	4.16	5.57	3.88

Tab. 5.5.: Test performance (classification error rate) as polarity classifiers. LSVM stands for Linear SVM. Surprise is our model mixing a recommender system and a linear regression machine. Finally, LSVM + Surprise mixes both predictions *a posteriori*.

The Surprise model provides lower performances than the strong SVM baseline. It is mainly due to the difficulty to control over-fitting in this context. However, the information learned by this model is different from the one of the SVM, as indicates the last column of table 5.5: the mixed models have better performance. This is the point of our study: to learn a different vocabulary, a surprise vocabulary.

5.3.4 Analysis of the weights

We propose here to compare the weights given to each term of the vocabulary by the two linear models: LSVM and the regression machine of our so called Surprise model. Both models attribute a weight to each term of the vocabulary. For the former model, top words are expected to express positiveness while bottom ones express the opposite. For the latter model, top words are used to increment the prediction of the recommender system, bottom ones to decrement it. Overall, extremely polarized words such as *disgusting* or *amazing* will often have similar weights. As we will see, the difference will be made on other terms, often related an unusual event that lowered or increased the satisfaction of the user.

We will focus on top and bottom words of the parameters of both the baseline and our disjoint model. Top words are referenced, for each dataset, in table 5.6. In each cell, words are ordered by non-increasing weight – as attributed in the weights vector of each model. As we can see, the top vocabulary is affected by the shift in formalism. Words like *upgraded* or *cares* for the Yelp dataset captures that the user – client of a business, typically a restaurant – received more attention than expected. On AmazonLiu, a dataset that contains many book reviews, it is interesting to see that top words defend the book w.r.t. other reviews that they find inaccurate. The same applies on the Movies dataset, that contains movie reviews. We are pleased to find this differences between the top words as it clearly indicates that our model take into account the context the user is in when writing the review. A similar behavior is observed when looking at the bottom words, presented in table 5.7. On Yelp, *rude* describes a bad service experience. On movie and book reviews, some users are so disappointed they want a *refund*. We believe that – in the grand scheme of centering studies around the users, concept is key.

Datasets	Top words baseline	Top words disjoint model
Yelp	delicious, excellent, perfection, pleasantly, downside, amazing, awesome, complaint, fantastic, perfect	**amazing**, **outstanding**, excellent, **cares**, best, blast, complaints, **upgraded**, incredible, thorough
AmazonLiu	refreshing, complaint, pleasantly, excellent, hilarious, awesome, bravo, rocks, funniest, succeeds	**haters**, best, **complaining**, refreshing, awsome, **misunderstood**, rocks, **reviewers**, **skeptical**, great
RateBeer	delicious, excellent, superb, wonderful, awesome, yummy, rjt, fantastic, lovely, perfect	**underrated**, **refreshing**, favorite, **surprised**, favorites, **enjoyed**, marks, rocks, lagers, girls
Movies	pleasantly, bbii, coulardeau, balian, quotable, complaints, refreshing, complacency, bedford, unfairly	awesome, awsome, **haters**, **complaints**, **complaining**, **complain**, great, funniest, **rocked**, refreshing

Tab. 5.6.: Top words per dataset for the baseline and our disjoint models. In order are the top 10 words of each model, ordered by non-increasing weight

Datasets	Bottom words baseline	Bottom words disjoint model
Yelp	worst, mediocre, meh, horrible, bland, disappointing, terrible, overrated, disappointment, poisoning	worst, horrible, **rude**, terrible, **awful**, mediocre, bland, **waste**, meh, poisoning
AmazonLiu	disappointment, yawn, worst, waste, uninspired, overrated, stinks, poorly, boring	waste, disappointing, disappointment, ugh, worst, overrated, boring, stinks, horrible, **refund**
RateBeer	mode, drain, meh, infected, disappointing, mess, disappointment, boring, mediocre, undrinkable	drain, **yuck**, **overrated**, disappointment, disappointing, undrinkable, **mess**, awful, drainpour, **worst**
Movies	kgharris, stinks, noooo, overrated, disappointing, disappointment, stinker, waste, disgraceful, lucasfilm	waste, **worst**, **refund**, **boring**, disappointing, disappointment, stinks, horrible, overrated, **defective**

Tab. 5.7.: Bottom words per dataset for the baseline and our disjoint models. In order are the bottom 10 words of each model, ordered by non-decreasing weight

5.4 Conclusion

As of [PL08], we know that for the classification of short English texts, linear models provide decent polarity classification models. But problems arise when enlarging the scope of the classifier. Generalization is one of them: how will the weights I learn a on particular dataset behave on another one ? With the abundance of user reviews available on the Internet, it is possible to use more training examples to improve generalization [Glo+11]. However, this is still limited to short texts, in the same language, on similar reviews – same website, same community. We believe that considering the context of a text is an important piece of the puzzle and have shown in this chapter that it was possible to add some context information to a linear classifier using recommender systems. Firstly, we have experimentally demonstrated that the more informative the profiles are, the better the classification is – in 5.2.1. Secondly, we conducted a series of experiments that confirmed that the information learned by recommender systems and linear classifier, in the context of polarity classification are different and can be combined efficiently (see section 5.2.2). Finally, we have proposed a way to use recommender systems to learn different weights for the vocabulary highlighting the surprise phenomenon in user reviews. This work is also presented in [Pou+14a] and [Pou+14b] for the polarity classification part and [] for the surprise part. Of course, this study could be followed in many ways. One is based on the idea of generalization and consists on validating the approach on datasets mixing reviews of various communities. Another is to drop the item profiles that are present through the recommender systems to use only user profiles. If these two steps are conclusive, then it would be possible to imagine extending the work to texts of various lengths: from tweets to full lengths blog posts.

6 Sparse event logs of user activities

Une journée type dans le coin: un facteur, un tracteur et rien.

— **Kamini**

Contents

For our final contribution, we consider another type of data generated by users: user authentication logs. These logs gather traces generated each time a user accesses a service such as servers and application databases of a company or identifies to work stations or websites. Compared to user reviews, authentication logs are massive, noisy and with a poor semantic. The useful information lies in the repetition over time of actions or sequences of actions. The characterization of stereotypes (of actions or sequences of actions) is an important task for the analysis of such data. Our contribution is a representation learning model that extracts such stereotypes in an interpretable way. We will experiment on the authentication logs of users of the Paris Metro. The database we use is provided by the STIF, *Syndicat des Transports en Île-de-France*, which is the political authority in charge of regulating transport in Paris and Ile-de-France. More than seven million people have subscribed to the Pass'Navigo, an identification card that grants unlimited access to the subway network in areas for which the user has subscribed. Our database gathers all authentications of theses users at station turnstiles.

With such data, it is possible to track user travels over time and to study their patterns. These patterns lead in turn to the extraction of robust user profiles. Our hypothesis is that the time and frequency of subway travels is a strong characterization of users. Our goal is to extract a set of typical activities and represent each user based on these activities. We define an activity or habit as a temporal event taking place at a certain time of the day, with possible repetitions during the week. For instance, taking the subway every working day at 8am is an activity.

Urban mobility is currently a field of intensive research, benefiting from the development of digital and physical tracking services that monitor users and provide huge amounts of data that can be used for various purposes. Firstly, in the context of development policies, such as [Bla+02], to propose key performance indicators for the development of sustainable transport systems. Many studies [Bro+06; Gon+08] are also interested in monitoring private vehicles, mainly cars, and use data from cellphone networks to track these trips and characterize their temporal scales. In [Son+10], the authors show that most of these private vehicle trips are predictable. They follow up on this work in [Wan+11] where they highlight the links between traffic profiles and profiles on social networks. Using data from cellphone networks as well, the recent study [Lou+14] analyses bottlenecks for each day of the week in the traffic of 31 Spanish cities. These data can also be used to detect anomalies in the network as in [Her10]. GPS data is a second type of data used widely in the literature, for example, in order to monitor taxis in Shanghai: [Pen+12] uses matrix factorization to characterize the behavior of taxi drivers. Due to the nature of the data, [Pen+12] cannot however track passengers over time. Other tracking techniques exist, such as using Bluetooth detector to monitor visitors of the Duisburg Zoo [Lie+12]. Similarly, in Paris, [Ran+13] extracts spatio-temporal clusters to analyze the usage of the Vélib' in Paris[1], which correspond to the trips of each individual bicycle but not of each user. The creation of cards identifying users [Gol02] allows large-scale analysis of trips with a fine-grained vision because it is possible to track users over time and to know precisely the time at which occurs each trip. This allows the authors to detect bottlenecks in transport networks in order, for instance, to propose alternative routes. [Cea+12] studies the spatial and temporal distribution of users in the London Underground. Similarly, [Foe+13] focuses on the prediction of future trips during the week days and for buses only. The purpose of [Foe+13] is to inform users in case of problems on the network. However, only a few studies are user-centered, even when working with timestamped user data, most work proceeds by data aggregation and characterizes the overall traffic rather than the behavior of individual users.

[1]A public bike sharing system.

In contrast to previous work, our contribution focuses on the discovery of a dictionary of activities and the representation of users in this space of activities. An activity is defined as an authentication at a given time of day, with a possible repetition during the week. As [Pen+12], our model uses a non-negative matrix factorization. We propose a new learning algorithm that, in addition to the non-negativity, imposes elements of the dictionary to have a certain shape, which corresponds to our definition of an activity.

6.1 Data representation

This first section discusses our choice of data representation, from the raw data, the user authentication logs to the final user profiles. These profiles are composed of three independent parts, each corresponding to independent authentication logs for the user. These logs are split into different sets based on the usage frequency of station. This split is necessary to push the analysis beyond the most recurrent trips, responsible of the vast majority of the traffic and on which most related work focuses. In our setting, it is possible to characterize infrequent trips as well.

6.1.1 User authentication logs

The authentication of user u accessing service s at time t is defined as a triplet (u, s, t). A log is a collection of such triplets, a user log gathers all authentications of one user over time. In our context (urban mobility), we study the usage of public transportation networks and the service that a user accesses is a subway station. On figure 6.1 is represented the log of an individual user, with all his authentications during the 13 weeks span of the dataset. The x-axis corresponds to time and the y-axis to stations (one line corresponds to a given station). Stations are ordered by usage frequencies, the most frequent ones are at the bottom and the less frequent ones are at the top. This log represents a typical user and two important facts have to be pointed out. Firstly, a limited set of stations, here two, are involved in the vast majority of the trips. We assume that these stations correspond to the ones closest to the user's home and to the user's workplace. Secondly, and as a consequence of the first point, a vast majority of the trips are made during working days and correspond to commuting. This has important implications, as we will see later on, on the ability of the model to correctly represent less frequent trips, occurring during evenings or the week-end for example. On figure 6.1, for the line of a given station, each spike corresponds to an authentication: the user takes the subway at this station. In the Parisian subway network, users only identify themselves using the Pass'Navigo when taking the train, not when exiting the network. This leads to an incomplete view of the trips as only the starting point of the trip is known. While knowledge about the destination would certainly be helpful, we believe that the information about the time at which each trip occurs provides enough information to strongly characterize users in terms of activities. For instance, trips corresponding to evening outs are strongly characterized by the time at which they occur.

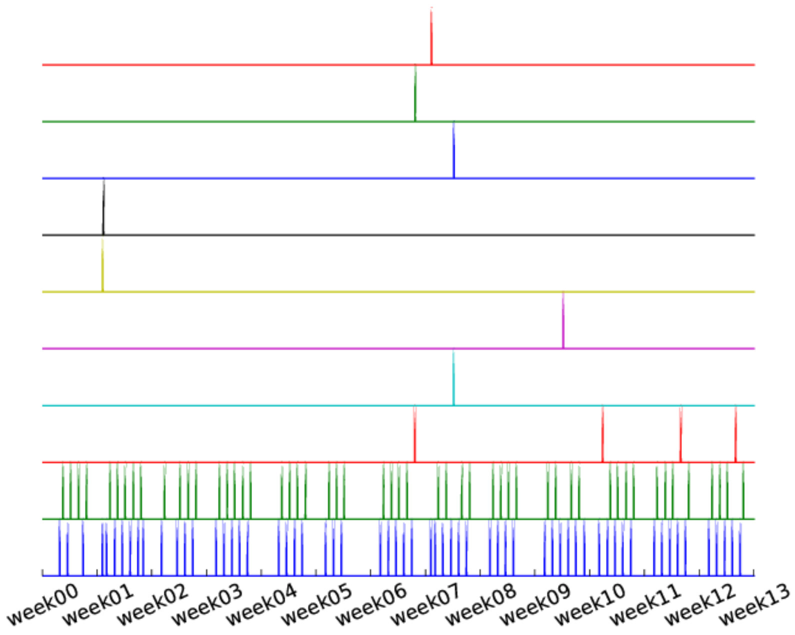


Fig. 6.1.: Individual user log: each spike is an authentication at a station, each station has its own temporal line. For most users, a small set of stations are frequently used. Here stations are sorted by usage frequency with the most frequent ones at the bottom.

6.1.2 Two temporal scales: day and week

Our definition of an activity implies two time scales: one for the time of the day at which a trip occurs and another for repetitions over the week. Our data is a collection of user logs, i.e. sets of triplets $L_u = \{(u, s, t)\}$, and we want to extract activities out of these logs and represent users in terms of these activities. We propose here a representation of these logs that consists of a daily and a weekly part and is robust to the individual variance inherent to the logs. This representation is based on an aggregation of the triplets into two vectors per user, one daily profile and one weekly profile. Both are based on time histograms, independently from the station s. For the daily profile, we divide the day into bins of 15 minutes, giving us $96 = 24 * 60/15$ features. For the weekly profile, we divide the week into bins of 2 hours, giving us $84 = 7*24*60/120$ features. Both are concatenated into one vector with $n = 180 = 96 + 84$ dimension. The contribution of a particular authentication in a user log to this representation can be visualized on figure 6.2. Independently of the station from which the trip originates, we construct histograms of the times at which they occur, using the two scales defined above. As a result, this representation records only temporal information, the spatial information is discarded, which corresponds to our definition of activities as temporal patterns.

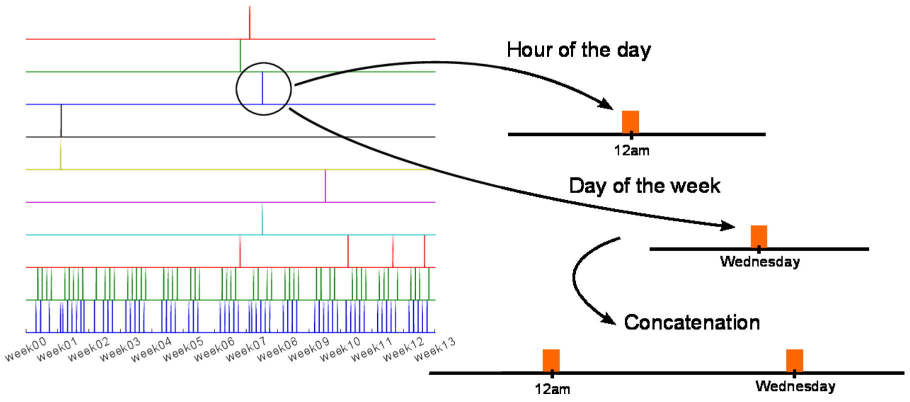


Fig. 6.2.: Computation of the histograms for a particular user. On the left are all the authentications of the user, as in figure 6.1. On the right, the bins corresponding to the time of the day and the day of the week of the selected authentication are incremented. At the bottom, the two histograms (day and week) are concatenated in one final vector.

However, some users take the subway more than twice a day when others take it once or twice a week. This high variability in usage would lead to an unbalanced representation of users. To overcome this we simply normalize user profiles and we choose to normalize the daily and weekly parts independently: each part must sum to one. This gives the same weight to both part when an overall normalization would favor the daily one. Profiles of multiple users are presented in figure 6.3. This representation is oriented so the x-axis corresponds to time. Thus, each row represents a user and columns represent the bins of the histograms. The darker a column is, for a given row, the more often this given user takes the subway at this time. To augment the contrast of the representation, a logarithmic color scale is used. The Parisian subway is closed between 2am and 6am (approximatively), thus there are no authentications during this period of time. The observation of figure 6.3 indicates that most of the traffic occurs during mornings and evenings of the workdays.

6.1.3 Frequency filters

This last observation hints that with the current representation, extracting infrequent activities will be difficult. This comes from the properties of the squared error loss for which, without regularization, models tend to overfit by focusing on signals with high energy values (here, highly frequent trips). As a consequence, we choose to split the original problem into three independent subproblems corresponding to different energy levels. This split is based on the observation already

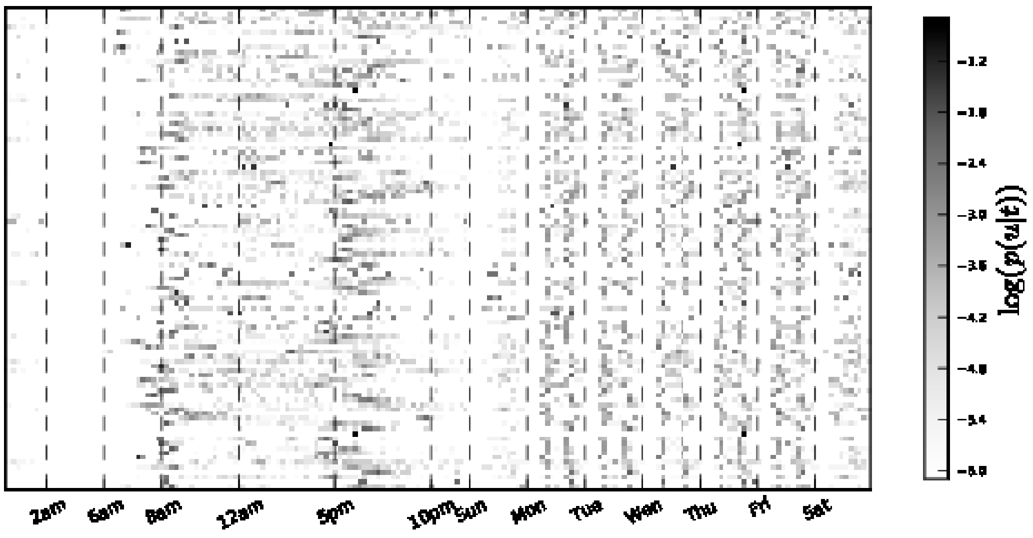
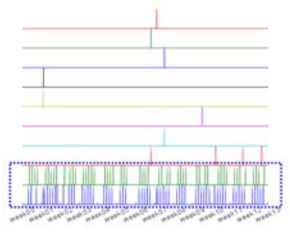
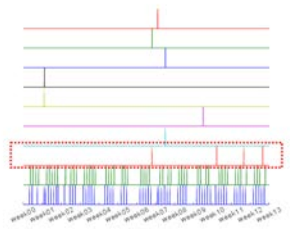


Fig. 6.3.: Raw user profile. Each line represents a user, the darker a column is, the more frequent it is for the user to take the subway at this time. Time is divided in two parts: time of the day or moment of the week. The former uses a 15 minute scale, the latter a 2 hours one. Most traffic occurs during mornings and evenings of workdays.

made on figure 6.1. Most of the traffic is generated by authentications at frequent stations where users takes the subway more than twice a week. Another intermediate part is due to recurrent trips due to sport clubs, visits to family and so on. The rest is due to infrequent trips happening less than once a week. This segmentation corresponds to our experience as users and has been validated with experts from the STIF. As a result, for a given user u, the stations where u have taken the subway are split into three groups, that we call frequency bands. The high frequency band $\mathcal{S}^{(high)}(u)$ groups together stations used more than twice a week by user u. The medium frequency band $\mathcal{S}^{(medium)}(u)$ groups together stations used less than twice a week but more than once a week by user u. The low frequency band $\mathcal{S}^{(low)}(u)$ groups together stations used less than once a week by user u. Figure 6.4 presents three visualization of the log presented in figure 6.1, each one with a dotted rectangle around stations of the corresponding frequency band.


(a)High-frequency stations


(b)Medium frequency band

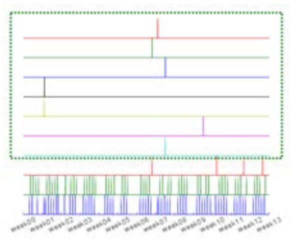

(c)Low frequency band

Fig. 6.4.: Stations of each user are separated based on the usage frequency of the user in three groups.

This segmentation of stations can be applied to user logs: logs are split in three frequency bands. Given u, a user, the low frequency band log of u, $L_u^{(low)}$, is the log of all trips starting at a station of the low frequency band $\mathcal{S}^{(low)}(u)$ for user u:

$$L_u^{(low)} = \{(u, s, t),\ s \in \mathcal{S}^{(low)}(u)\} \tag{6.1}$$

We define in the same way the medium and high frequency band logs, $L_u^{(medium)}$ and $L_u^{(high)}$. Using these three user authentication logs per user, we can now compute three vector representations for each user. Each representation is computed as before using the user log of one energy band. Grouping these vectors in matrices, we can then define three data matrices, one per frequency band, $\mathbf{X}^{(low)} \in \mathbb{R}^{n\times m}$, $\mathbf{X}^{(medium)} \in \mathbb{R}^{n\times m}$ and $\mathbf{X}^{(high)} \in \mathbb{R}^{n\times m}$. We denote m the number of users. The i-th column of each matrix is the vector representing the i-th user in the corresponding frequency band.

Figure 6.5 displays the profiles of the same users as in figure 6.3 but with the frequency filter. Matrices are presented in the same way and from bottom to top are $\mathbf{X}^{(low)}$, $\mathbf{X}^{(medium)}$ and $\mathbf{X}^{(high)}$[2]. As for figure 6.3, we augment the contrast through a logarithmic color scale. As expected, the $high$ frequency band contains commuting patterns: a validation in the morning and one in the evening, often with really low variability, and repetitions on working days. The $medium$ frequency seems to catch late morning departures, lunch break and evening activities. The low has a high variance for most users, which corresponds well to occasional activities.

6.2 Extraction of activities through matrix factorization

In the previous section, we have built a representation of the data where each user is represented as three independent vectors, one per frequency band. Our goal is to extract activities out of this representation and represent users in terms of these activities at the same time. We will proceed independently on each frequency band and we use a non-negative matrix factorization for this purpose. Matrix factorization provides a framework to simultaneously learn a dictionary and a code as described in chapter 3. Here the dictionary is the set of activities and the code is the repartition of activities among users. Moreover, the non-negativity constraint is meaningful here and leads to an interpretable solution. However it is not clear whether elements of the dictionary will satisfy our definition of an activity: a temporal event occurring at a certain time of the day, with a possible repetition during the week. In this section, we propose a factorization algorithm that forces elements of the dictionary to match this definition.

[2]Figures represent the transpose of the matrices so the x-axis is the time axis, which is a more natural representation for visualization purposes.

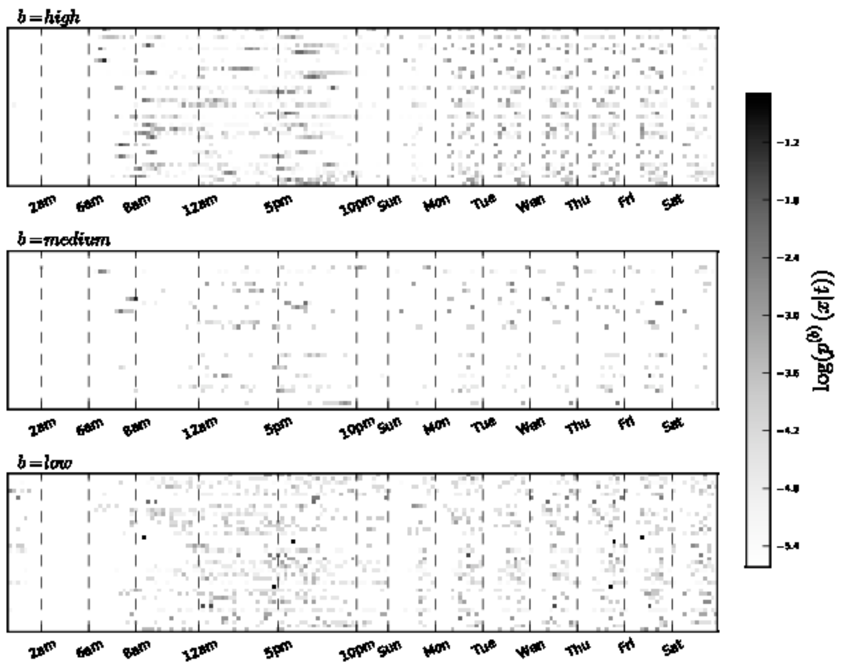


Fig. 6.5.: Representation of each data matrix: one per frequency band. As in figure 6.3, each row represents a user and the i row of each plot corresponds to the same user. Each column is a time bin, counting the number of authentication at stations of this frequency band, for each frequency band. We use the transpose so that time is along the x-axis, which is more natural.

6.2.1 Problem formulation

We denote $\mathbf{X}^{(b)} \in \mathbb{R}^{n \times m}$ the data matrix of frequency band b, as presented in 6.1.3. The i-th column of $\mathbf{X}^{(b)}$ is the temporal profile of the i-th user, described by n features, for the frequency band b. We will learn one non-negative matrix factorization per frequency band. The same algorithm is used independently for all bands, thus we now drop the (b) subscript.

We wish to extract a dictionary of k typical behaviors $\mathbf{D} \in n \times k$ as well as the coordinates $\mathbf{H} \in \mathbb{R}^{k \times m}$ of the m points in the k dimensional space defined by the k columns of $\mathbf{D}$. Hence the reconstructed user profiles are obtained by computing the product $\hat{\mathbf{X}} = \mathbf{DH}$. We use a non-negative matrix factorization with a regularized mean squared error for the reconstruction loss. We assume that each user has to be described by a few typical behaviors and thus constrain $\mathbf{H}$ to be sparse, as in [Hoy02; Mai+10]. As a result, the regularized loss that our model optimize is:

$$\mathcal{L}(\mathbf{X}, \mathbf{D}, \mathbf{H}) = \frac{1}{m} \|\mathbf{X} - \mathbf{DH}\|^2 + \lambda |\mathbf{H}| \tag{6.2}$$

The non-negativity constraint is written:

$$\mathbf{H} \geq \mathbf{0}, \mathbf{D} \geq \mathbf{0} \tag{6.3}$$

As in [Hoy02; Mai+10], columns of $\mathbf{D}$ are normalized to avoid unreasonable solutions obtained by arbitrarily transferring the magnitude of $\mathbf{H}$ to $\mathbf{D}$. The nature of the elements of the dictionary, columns of $\mathbf{D}$, is a bit peculiar in our setting: each column is divided in two parts, a daily one and a weekly one. Since we normalized the columns of the data matrix (user profiles) $\mathbf{X}$ so that the daily and weekly part independently summed to one, we do the same thing for the dictionary. We denote n_{day} the index such that feature i is from the daily profile if $i \leq n_{day}$. The normalization of both parts of each dictionary element is written as:

$$\forall j, \sum_{i=1}^{n_{day}} \mathbf{D}_{ij} = 1 \text{ and } \forall j, \sum_{i=n_{day}+1}^{n} \mathbf{D}_{ij} = 1 \tag{6.4}$$

As we said above, with such a setting, the elements of the dictionary may not match our definition of activities: events happening at a certain time in the day with a repetition during the week. We want each column of $\mathbf{D}$ to identify a specific and easily interpretable activity so that a user could be described as a simple mixture of these events. Such a typical event would be: recurrent trip on weekdays at 8am (interpretable as going to work), recurrent trip at 1pm on Mondays or occasional trip at 11pm on Thursdays. To force elements of the dictionary to match this definition, we will force the daily part of each column of $\mathbf{D}$ to have a Gaussian like distribution with one peak corresponding to the time of the day at which the activity associated to this column vector occurs. We control the width of the peak using an hand-fixed ratio $\frac{\delta_t}{2\sigma}$ and denote i_{peak} the index of the peak in the vector. The constraint we impose on the dictionary is written as:

$$\forall j,\ \forall i \leq n_{day},\ \mathbf{D}_{ij} \approx \exp(-\frac{\delta_t^2 (i - i_{peak})^2}{2\sigma^2}) \tag{6.5}$$

For our application, as it is not physically possible to authenticate twice in less than a few minutes, we set δ_t to 15 minutes.

6.2.2 Training algorithm

The learning algorithm we propose is based on the algorithm of [Hoy02] and that is presented in section 3.2.3. We adapt it to our constraint on the shape of the elements of the dictionary. Optimisation is performed by a batch gradient descent. The dictionary is updated with a projected gradient algorithm: the projection ensures that constraints are satisfied. In our algorithm, algorithm 8, the projections associated to the non-negativity and normalization are always performed, thus the elements of the dictionary remain non-negative and well normalized. However, the projection associated to the shape constraint is performed every n_G iterations, with n_G hand fixed to 200 for our experimentations. This is a hard projection, that erases great proportions of the signal of each element of the dictionary. Thus, applying it more often may hinder the convergence of the matrix factorization. The code matrix is updated with multiplicative update rules.

```
**Data**: $\mathbf{X}$, $\mathbf{D}_0$, $\mathbf{H}_0$, $\lambda$, $n_{epochs}$, $\mu$, $n_G$
**Result**: $\mathbf{D}$, $\mathbf{H}$
$\mathbf{D} \leftarrow \mathbf{D}_0$ and $\mathbf{H} \leftarrow \mathbf{H}_0$;
**while** $e \leq n_{epochs}$ **do**
    $\mathbf{D} \leftarrow \mathbf{D} - \mu(\mathbf{X} - \mathbf{DH})\mathbf{H}^t$;
    $\mathbf{D} \leftarrow \max(\mathbf{D}, \mathbf{0})$;
    Normalize daily and weekly profiles of $\mathbf{D}$ to sum to 1;
    **if** $e \mod n_G = 0$ **then**
        $\forall j, \mathbf{d}_j \leftarrow \mathbf{d}_j \odot \exp(-\frac{\delta_t^2 (i - i_{peak})^2}{2\sigma^2})$
    **end**
    $\mathbf{H} \leftarrow \mathbf{H} \odot \mathbf{D}^t\mathbf{X} \oslash \mathbf{D}^t\mathbf{DH}$
**end**
```

Algorithm 8: Batch gradient based algorithm to solve the constrained matrix factorization. It alternates projected gradient updates on $\mathbf{D}$ and multiplicative update rules on $\mathbf{H}$. The projection associated to the shape constraint is done only every n_G iterations.

We will derive here a sketch of convergence proof. Our goal is not to obtain analytical bounds on the convergence speed but to claim that imposing shape constraints as we do leads to convergence. The overall convergence of the algorithm without the shape constraint is well known. Multiplicative update rules are known to converge as shown in [LS01]. The work of [GZ05] shows that the convergence rate can be increased in practice but confirms the convergence of multiplicative update rules. For gradient updates, [Hoy02] provides an experimental justification of the convergence. Theoretical arguments are provided in [Lin07] that propose to update both parameters using gradient update rules and to increase convergence rate by selecting optimal parameter at each step instead of single updates. As described earlier, the projection associated to the shape constraint is done only every n_G iterations. Thus, we propose to see the algorithm as follows. First, we use an algorithm that is known to reach convergence, without the shape constraint to find good parameters $\mathbf{H}$ and $\mathbf{D}$. We then apply the constraint and use the parameters as initial values for the basic algorithm. It is basically a warm restart. Applying the constraint too often may hinder convergence, but we found that, in practice, applying it every 200 iteration is a good compromise.

6.2.3 Analysis of extracted dictionaries

We will now analyze the patterns extracted by our algorithm on a dataset containing around 80 million trips, 600k users and 300 stations. We have implemented our models using Python and Numpy [VDW+11] and we have run the computations on commodity machines with 8 cores (3.07GHz) and 16GB of RAM. The computations took around 10 hours.

As we said early, algorithm 8 is applied independently on each frequency band. Thus, we obtain three dictionary matrices: $\mathbf{D}^{(low)}$, $\mathbf{D}^{(medium)}$ and $\mathbf{D}^{(high)}$. Each of them contains $k = 100$ basic elements that represent activities. These dictionaries are represented in figure 6.6. For this visualization, the columns of the matrices are ordered according to the hour of the highest peak to exhibit that each different band focuses on different parts of the day. The dictionary associated to the high frequency band, on the left of figure 6.6 focuses on mornings and workdays, which was expected as we observed from the data that most trips occur during mornings or evenings of workdays. The benefit of our approach is that our analysis can got beyond these frequent trips when related works, as [Foe+13], focus only on these. Thanks to our frequency filtering, we have two additional dictionaries. The one for the medium frequency band focuses on afternoons and evenings of the workdays and of the Saturdays. The one for the low frequency band focuses mainly on evenings of all days, which corresponds to what was expected: this dictionary provides a characterization of infrequent (recreational) activities. As for other visualization, figure 6.6 is oriented so the x-axis is the time: each row corresponds to one element of a dictionary. An interesting behavior emerged in the $high$ frequency dictionary. Even though basic elements are constrained to have only one peak, many have two. This is due to the high regularity in commuting pattern: they are based on office hours that are mostly fixed and part of the $high$ frequency band. This phenomenon is not observed on other bands.

6.3 Application to station clustering

Our non-negative matrix factorization algorithm provides, in addition to the dictionaries, code vectors that are the representation of each user in the space of activities. We will describe this representation first. We believe that it provides a strong characterization of the users. We would like to use this representation to predict personal information about the users, such as the socio-professional category. We claim that the time at which users take the subway is representative of their social status. Firstly, the user's job influences directly the user's commuting pattern and the dictionary of the high frequency band is able to encode many different patterns. Secondly, social groups also impact the number of recreational activities of users, which impacts in turn the encoding of users for the low and medium frequency bands. However, personal information on the users is not available, for obvious privacy reasons. We propose here a surrogate evaluation by clustering stations.

6.3.1 Users as bag of activities

The previous section analyzes the dictionaries extracted by our algorithm, one per frequency band. We will now discuss briefly the code that represents each users: we aim at obtaining bag of activities for each user, the activities being the elements of the dictionary. The code vectors representing users can be seen as quantifiers of the repartition of activities for each user. For a given user and a given frequency band, the value of the i-th entry is directly linked to the ratio of authentications made by this user that corresponds to the i-th activity.

Let us consider a practical example. Say a user only takes the subway on mornings of workdays at 8am and on evenings of workdays at 6pm. Say that the extracted dictionary for the high frequency band contains these two activities (8am on workdays and 6pm on workdays) as its 10th and 57th activities. Then, the code vector of this user for the high frequency band will only have two non-zero entries, the 10th and the 57th, that should have equivalent magnitudes. Moreover, code vectors for the two other frequency bands are zero, as no authentication happens for this user in these bands. The direct implication of such a representation is that this user will be strongly separated from users having authentication in the low or medium frequency band, on the one hand, and from users having different commuting times, on the other hand. As the dictionary provides many different activities, we can obtain a fine-grained representation of users.

Matrix (% of non-zero entries and std)	*low*	*medium*	*high*
Data sparseness (std)	16.98 (0.38)	7.97 (0.27)	16.88 (0.37)
Dictionary sparseness (std)	51.15 (0.50)	52.41 (0.50)	51.93 (0.50)
Code sparseness (std)	9.23 (0.29)	4.33 (0.20)	5.46 (0.23)
Important activities (std)	4.38 (0.20)	1.94 (0.14)	1.68 (0.13)

Tab. 6.1.: Sparseness measures on the *low*, *medium* and *high* frequency bands. The first row reports the average number of non-zero entries per user in the raw profiles. The second row reports the average number of non-zero entries per activity in the dictionaries. The third row reports the average number of non-zero entries per user in the code matrices. The fourth and last row reports the number of important activities per user, a quantification, detailed in the text, of the number of significant activities per user.

The code representing each user is typically sparse as reports table 6.1. For the first row, the sparseness of the data, it can be observed that the high and low frequency bands have profiles with similar sparseness. However, the reason is different for each band. The high frequency band correspond to heavy traffic concentrated on the mornings (and evenings in a smaller proportion) with many repetitions during the week. Typically, the weekly part of user profiles in the high frequency band is not sparse. For the low frequency band, this is simply due to the high variance in the time of trips done in this frequency band that correspond to infrequent activities, scatter across the week, explaining the (relatively) low sparseness. In average, for all the frequency bands, out of the 100 activities per frequency band, a user is described by less than 10 in average. Also, among these non-zeros entries, some have a low magnitude. We define the number of important activities as the number of top-magnitude entries necessary to account for 90% of the total magnitude of the code vector. It is a concept similar to the ratio of explained variance for PCA. The average number of important activities are reported on the last row of table 6.1. This confirms our interpretation of the decomposition. On the one hand, the high frequency band focuses on commuting patterns and can be described by one or two activities (going to work, going back home). On the other hand, the low frequency band gathers many infrequent activities, justifying the greater number of activities necessary to explain the behavior of a user.

6.3.2 Projection of user behaviors on stations

The spatial information has been discarded when computing the user profile for each frequency band. Thus, we currently have no information to describe stations. We will use the user logs once more to build a representation for each station and for each frequency band. We project the representation of each user to the station he uses, per frequency band. Thus a user visiting a particular station for the high frequency band will transfer his code, his repartition of activities, for this band to the station. As a result, each station is represented by average repartitions of activities per frequency band and reflects the behaviors of the users. Also, using the dictionaries, we can project the representation in the activity space back to the temporal space. We will use this in the following section to analyze the traffic of each station cluster.

Analyzing the spatio-temporal usage patterns requires establishing connections between a temporal event and a station. This is not straightforward and we propose below, for each frequency band some possible interpretation. The key concept is that a validation corresponds to leaving a place, not going to this place.

high This bands corresponds to users that frequently take the subway at these stations. Most of the traffic here is due to commuting patterns. As so, interpretation is easy for this band: it corresponds to people leaving or working nearby. Activities related to this frequency band are leaving home or work.

medium This band corresponds the stations where users take the subway once or twice a week. It corresponds to recurrent activities such as meals with family members or friends nearby or a club or regular activity. However, as only the origin of trips is known, it means that the user is leaving after this activity.

low It corresponds to users that takes this station less than once a week. There are no regular patterns here. It could be a meeting elsewhere or a restaurant. Once again, the important thing is that we detect the user when he leaves this activity to go somewhere else.

6.3.3 Station clustering

Each station is represented by three vectors, one per frequency band. Best results were obtained with a simple concatenation of this vectors. Each station is represented by one vector concatenating the low, $medium$ and $high$ latent representations. We use a k-means algorithm on this representation to cluster stations in coherent groups that share similar activities distributions.

Figure 6.8 represents the map of the subway stations in Paris where each station is colored (and shaped) according to its cluster. Some geographical patterns clearly emerge. We have reported on figure 6.9 the difference between the average traffic and the reconstructed traffic of each centroid: a positive spike means more traffic in this cluster than in the network. Finally, to given so background material on the sociological context of Paris, figure 6.7 reports the geographical distribution of the incomes in the region. The data for this is provided by the INSEE, a political institution in charge of population surveys in France. We will now provide an analysis of our cluster. It is interesting to compare this clustering presented in figure 6.8

to the sociological geography of the city in figure 6.7: our clustering matches the geographical distribution of incomes. For instance, western suburbs are wealthier than eastern ones with the exception of Vincennes and, in our clustering, Vincennes is part of the western cluster. Overall, there are two inner clusters describing the center of the city, one belt-shaped cluster around this center and the separation of the western and eastern sub-urban regions around Paris.

First the touristic center in green triangles (c03) of Paris with *Champs Élysées* and *Concorde*, the Louvre Museum, the Garnier Opera, *Notre Dame* and the *Sacré Cœur* and also, noticeably, the *Parc des Princes* stadium are in the same cluster. Second, the belt-shaped cluster, here in yellow squares (c02), corresponds to the limits between Paris and the surronding cities. The city limits are marked by gates (*Porte* in French) as a reminder of the gates piercing the old fortified walls of the medieval Paris.

And last but not least, the clustering opposes the posh western sub-urban regions of Paris, in dark blue squares (c05) to poorer eastern sub-urban regions in red circles (c01). The similar distinction can be observed on the data of the INSEE on figure 6.7, reporting the repartition of incomes in *Île-de-France*. The distinction, based on temporal patterns, is interesting as the users might have the same patterns since they are at the same distance from the city center. So the distinction goes beyond the simple geographic explanation and touches a sociological repartition of work hours.

We will now analyze the temporal profiles of each centroid. We have reconstructed it by projecting the bag of activities representation of each centroid back to the temporal domain using our dictionaries. The difference between each cluster have small magnitude with respect to the average traffic load. Thus, we choose to focus on these differences by reporting on figure 6.9 the delta to the average traffic load for each centroids. A positive spike means more users authenticating themselves at any station of the cluster than in average over the network. A negative spike is the opposite: a deficit of users compared to average load. Each row corresponds to a centroid and columns are grouped by pairs: each pair corresponds to a frequency band (from left to right: high, medium and low) and within each pair the first column is the daily profile, the second is the weekly profile. Each cluster centroid is colored as the corresponding cluster in figure 6.8. As only the check-in authentications are available to us, the high frequency band corresponds to the habit of the people living or working near the stations of the cluster while the medium and low frequency bands correspond to people check-in to travel elsewhere, meaning that they came to the stations of the cluster for some periodical activity like pubs, restaurants, theaters and so on and are now leaving. The last line is the belt cluster that corresponds to the average behavior of standard Paris dwellers which is coherent with the fact that stations composing this cluster are at the limit of the city.

The first line and third lines are close one to other. The former corresponds to the touristic center of Paris which is sensible as it is characterized by a lack of check-ins in the morning for no working class is living there. Once again the temporal pattern is linked to a sociological repartition. The latter contains the big clusters corresponding to train stations. People working in sub-urban regions where the subway network is not present typically take the train to one hub and then finish their journey with the subway. This explains the main difference in the high frequency band between the two clusters: the peak around nine in the morning. It is also noticeable that these two clusters are the only ones having people departing after infrequent activities, as can be seen on the low frequency band. The second and fourth lines correspond respectively to the western and eastern sub-urban regions around Paris. As said earlier they are mainly residential areas with the western part being wealthier than the eastern one. This is confirmed by the night part of the medium and low frequency bands, in contrast to the center clusters, few people are leaving these clusters after some episodic activities. The phenomenon of a peak in the medium band correspond to the activity of leaving for work from a station that is not the main home of a user and is typical of sleepovers. Finally the commuting patterns of the two clusters is different. The model is able to extract fine grained representations and distinguish the clusters as they do not commute at the same time.

6.4 Conclusion

This is the first user-centered study on the Parisian subway using authentication logs. We have provided a framework to build out of these logs robust profiles that strongly characterize users. To do so, we have proposed an adapted matrix factorization, demonstrating once again the plasticity of these representation learning methods. The extracted profiles, once projected on stations, lead to a clustering that is linked to socio-demographic patterns. The work we conducted on urban mobility lead to a publication [Pou+]. Many perspectives arise from this study. Firstly, we integrated in the representation the notion of repetition during the week by using daily and weekly parts. This has provided a simple way to extract interpretable activities in the dictionary using matrix factorization as event occurring at a certain time of the day repeated during the week. However, it may be possible to use models with memory capability such as recurrent neural networks [Gra12] or adapted matrix factorization [BG09] and to work on days only. Secondly, we have proposed a user-centered analysis by extracting profiles that characterize the users and seem to be linked to socio-demographic patterns. It could be interesting to use external data, such as information about places around stations extracted from services like Google Map, to validate our clustering of stations. Finally, other uses of the extracted profiles are possible. Comparing the logs of a day to what is expected for such a day based on extracted profiles can be used as an anomaly detection technique. It is similar to the idea of [Mai+08] that uses the reconstruction error of matrix factorization to discriminate examples.

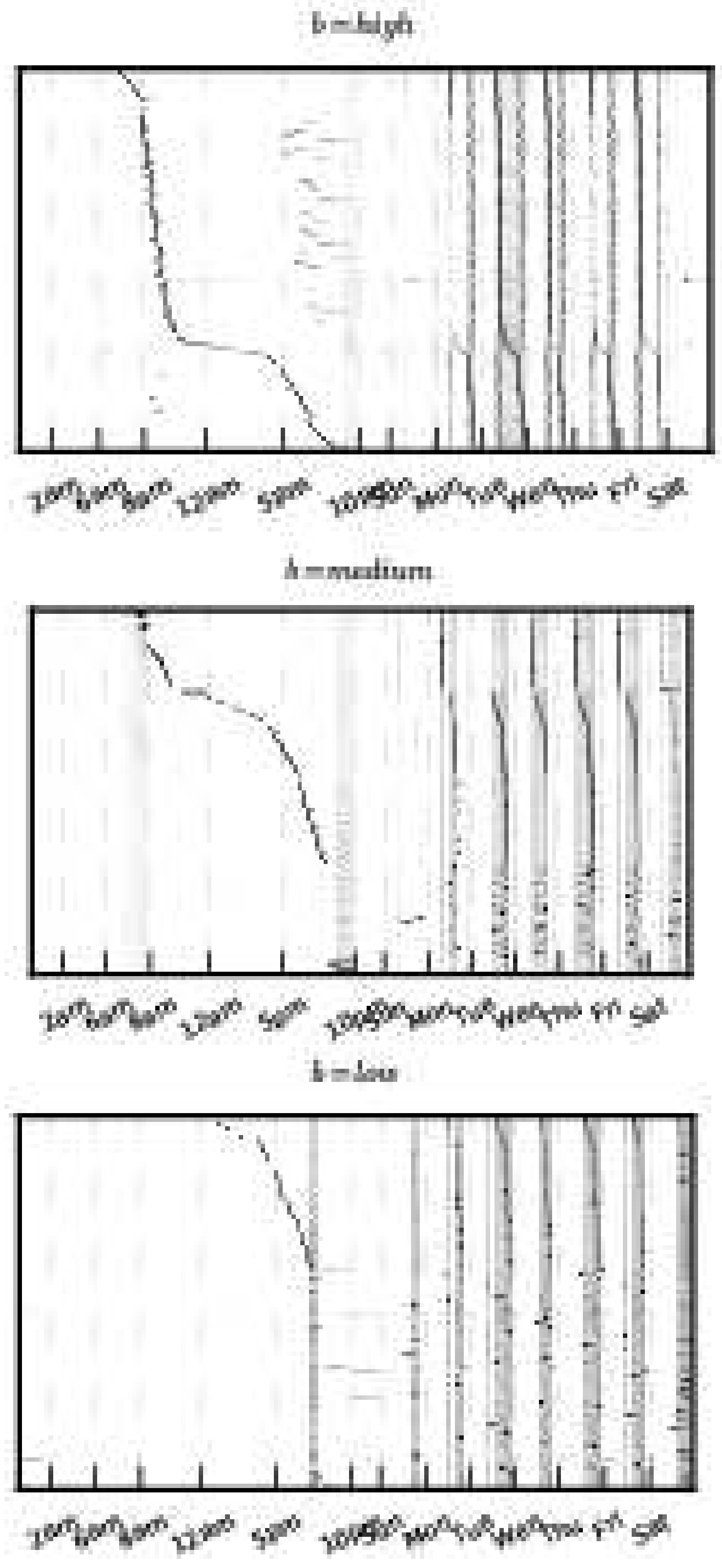


Fig. 6.6.: Dictionaries extracted using algorithm 8, one per frequency band. From top to bottom are $\mathbf{D}^{(high)}$, $\mathbf{D}^{(medium)}$ and $\mathbf{D}^{(low)}$. The scale is common to all plots. Dictionaries are transposed to the x-axis is the time. Each row is a basic element of the associated frequency band. Basic elements are sorted by non-decreasing time of the highest peak.

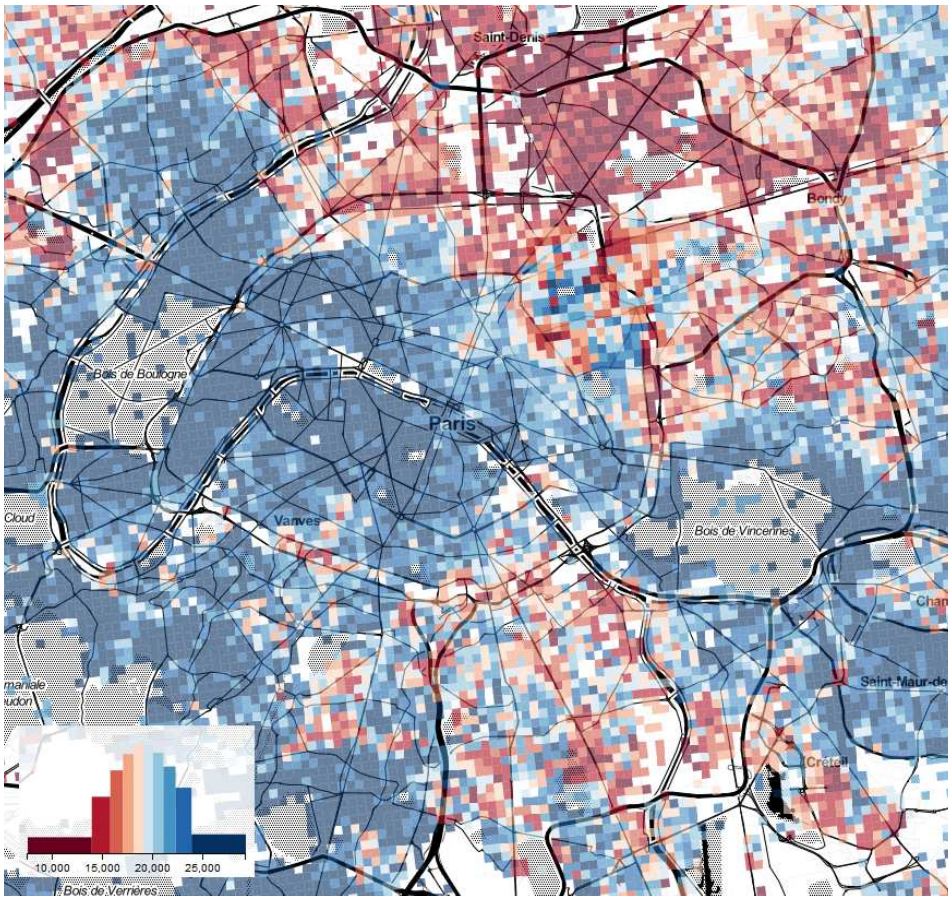


Fig. 6.7.: Income per household in the Ile-de-France. Clusters extracted by our analysis and presented in figure 6.8 align with the geographical distribution of income.

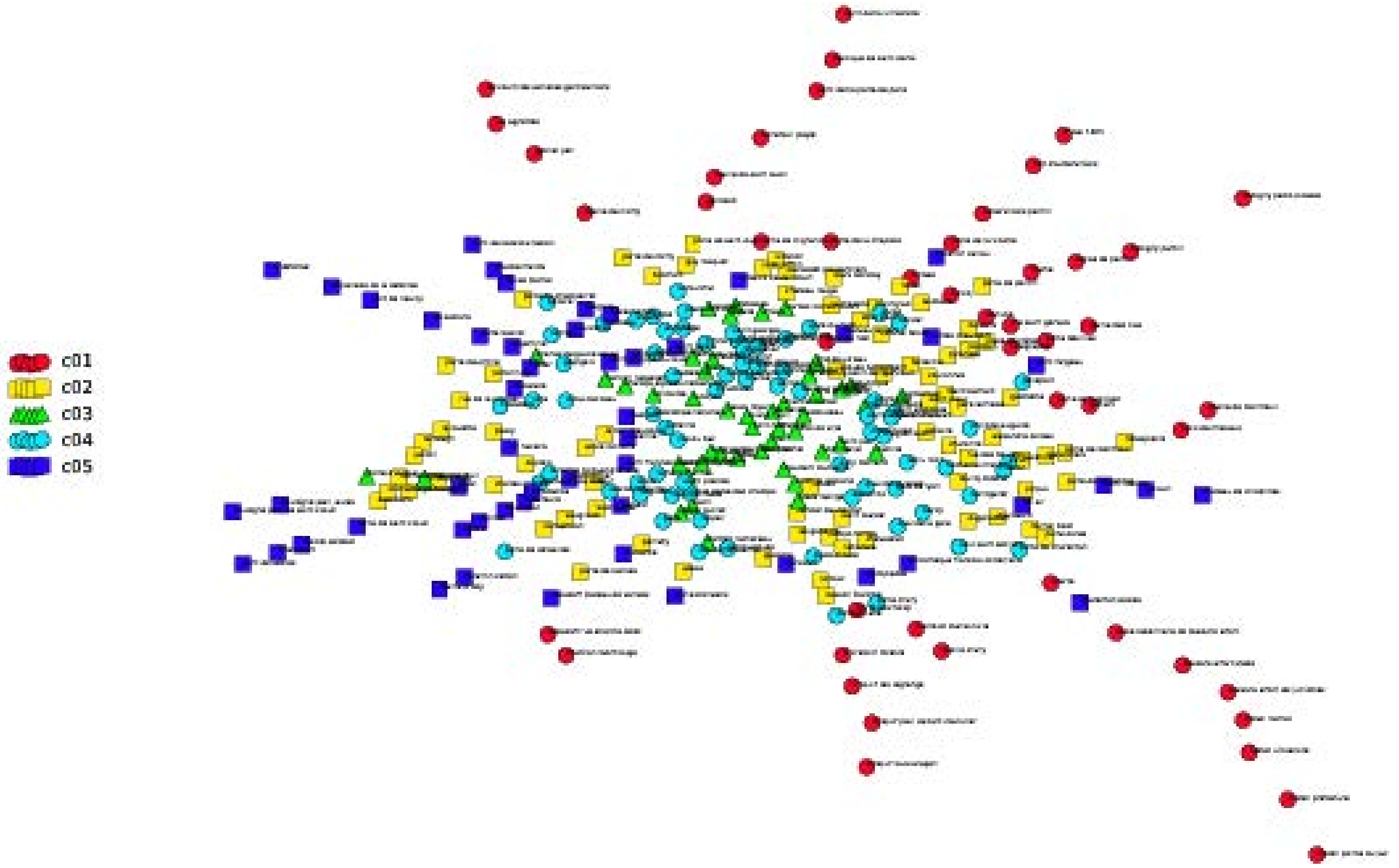


Fig. 6.8.: Map of Paris subway stations with colors coming from our clustering.

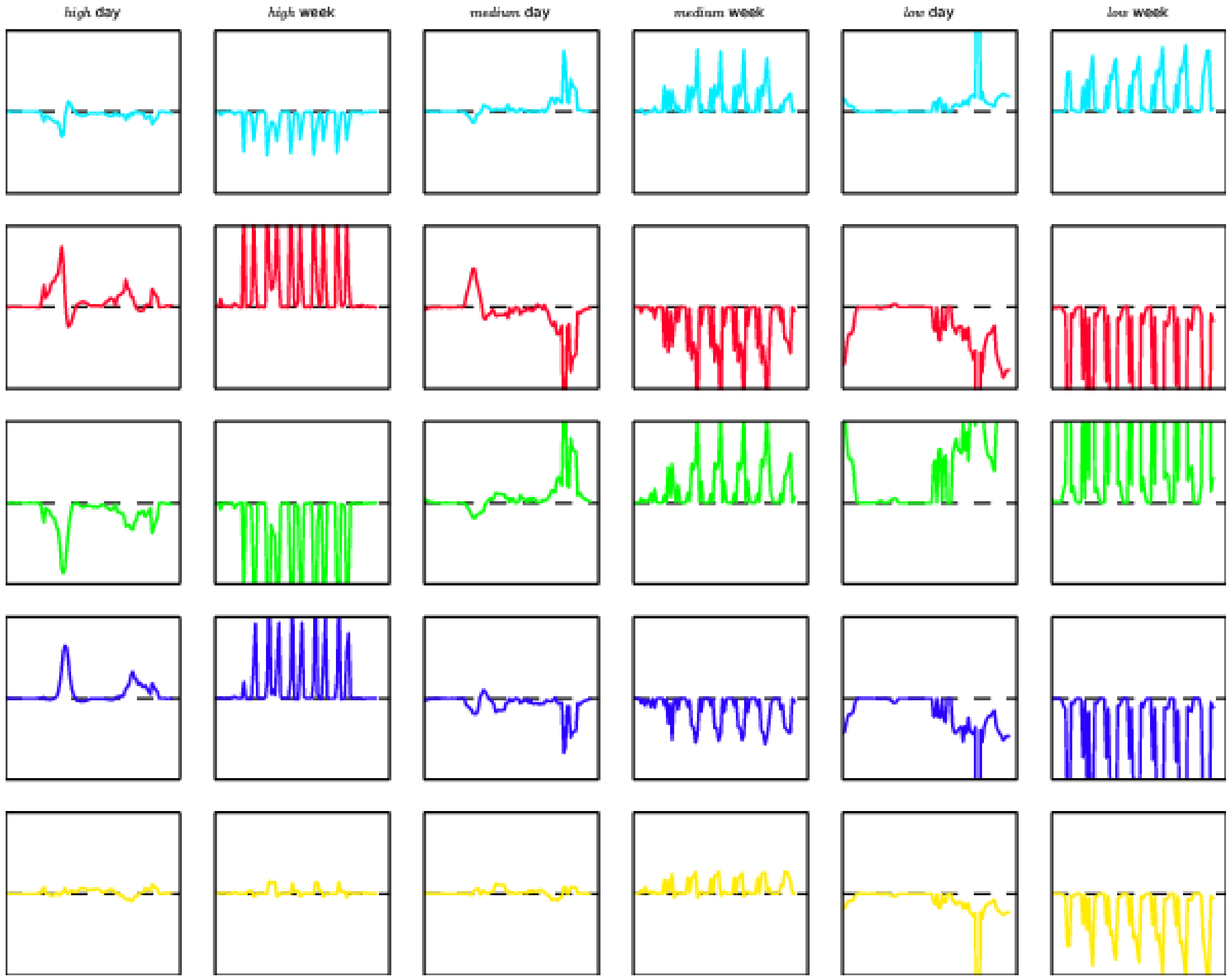


Fig. 6.9.: Temporal representation of the centroids of the clustering in Figure 6.8. The x-axis is the time.

Conclusion 7

We have studied, throughout this thesis, various ways to represent users. This problem is at the crosspoint of two research axis of study: representation learning and user profiles.

Representation learning proposes to replace the laborious task of hand-crafting features by machine learning algorithms. Traditional approaches rely heavily on such hand-crafted features; indeed, models operating on relevant features have good performance for classification or regression tasks. However it is a tedious and time consuming task. We have seen two families of representation learning methods: neural networks and matrix factorization, both can be used on very large datasets.

Neural networks are complex models with an extremely high variance: they are very sensitive to small perturbations in the datasets [Sze+13]. We have presented neural networks, in chapter 2, using the deterministic framework of multi-layer perceptrons. For these stacked models, each layer learns a representation of the data that disentangles it for the next layer. Recent breakthroughs in deep neural networks, especially for signal processing tasks such as image classification [Cha+14] or handwriting and speech recognition [Gra12] and text representations [Col+11; Mik+13; LM14] have been made possible thanks to the availability of large datasets and efficient computers.

Matrix factorization provide a natural framework to code each example of a dataset as a combination of basic elements learned from the data, gather in a dictionary. While most methods of machine learning focus on learning either the dictionary or the code matrix, matrix factorization learns both at the same time. This is a difficult problem as it involves many parameters and matrix factorization has quite a strong variance as well as neural network. One common way to limit the impact of the variance is to impose constraints on the dictionary and/or the code. These constraints, the most common ones are non-negativeness [LS01] and sparseness [Hoy04], deal with the variance by adapting the factorization to the nature of the data as well as providing interpretable representations of the data.

These representation learning techniques provide a gateway for user modeling. We started by considering the case of recommender systems in chapter 4 where we demonstrated the benefits of considering both the ratings and texts of user reviews to learn more informative profiles. We have also proposed an extension of the traditional framework of recommender systems that consists in generating a text in addition to the rating prediction task. This led us to study the impact of user profiles on the classical task polarity classification in chapter 5. We have shown that it was possible to jointly train a recommender system and a classifier to detect words that highlight the surprise of a user given a product. Finally, we built on the experience gained from the work with user reviews to adapt and apply a classical tool – namely non-negative matrix factorization – to learn user profiles in the context of subway usage in chapter 6. We have demonstrated that it was possible to constrain the factorization so as to extract habits out of the data and explain each trip made by a subway user using one of these hidden habits. This work however can be easily adapted to various kind of user event logs as it only requires only logs of the form (user, event, time).

We believe that the mass of available data tends to drown users. Correctly apprehending user profiles to provide a personalized access to information is the way to overcome this challenge. We have shown the impact of enriching user (and item) profiles of recommender systems both to improve the performance of existing models and to provide a more personalized recommendation by generating short summaries of other reviews. Additionally, we provided a method that extract robust profiles out of noisy data, the authentication log of users. This study takes place in the physical world, using authentication of users at subway station turnstiles and offers new opportunities, ranging from the recommendation of social events in the city to personalized real-time information about the service status.

For future work, we have considered two main options. The first is the proposal of a unified model that could bind together the tasks of chapters 4 and 5. We would like to use new representation models as SkipGrams or C-Bow networks to do so by adding additional look-up tables for the users, items and using supervision to take into accounts the available ratings. The second is to look at the dynamics of the user profiles. This can be tackled in many ways. For the work on the subway usage for instance, we may want to extract regular pattern and use these to predict expected traffic or trips and detect when the ground truth diverges from what we predict. Adapting models to take time into account is far from trivial. Both matrix factorization and neural network already propose some algorithms. Going deeper in this direction is certainly an interesting perspective.

Bibliography

[Aga+11] Deepak Agarwal, Bee-Chung Chen, and Bo Pang. „Personalized recommendation of user comments via factor models“. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, pp. 571–582 (cit. on pp. 48, 59).

[Ama+96] Shun-ichi Amari, Andrzej Cichocki, Howard Hua Yang, et al. „A new learning algorithm for blind signal separation“. In: *Advances in neural information processing systems* (1996), pp. 757–763 (cit. on p. 37).

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. „Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions“. In: *Knowledge and Data Engineering, IEEE Transactions on* 17.6 (2005), pp. 734–749 (cit. on p. 52).

[AU07] Massih Amini and Nicolas Usunier. „A contextual query expansion approach by term clustering for robust text summarization“. In: (2007) (cit. on pp. 5, 69, 72).

[Bac14] Francis Bach. „Beyond stochastic gradient descent for large-scale machine learning“. In: (2014) (cit. on pp. 16, 58).

[Bar+98] Andrew Barron, Jorma Rissanen, and Bin Yu. „The minimum description length principle in coding and modeling“. In: *Information Theory, IEEE Transactions on* 44.6 (1998), pp. 2743–2760 (cit. on p. 25).

[Bat92] Roberto Battiti. „First-and second-order methods for learning: between steepest descent and Newton's method“. In: *Neural computation* 4.2 (1992), pp. 141–166 (cit. on p. 12).

[Bel+90] Richard K Belew, John McInerney, and Nicol N Schraudolph. „Evolving networks: Using the genetic algorithm with connectionist learning“. In: *In*. Citeseer. 1990 (cit. on p. 12).

[Ben+06] Emmanouil Benetos, Margarita Kotti, and Constantine Kotropoulos. „Musical instrument classification using non-negative matrix factorization algorithms“. In: *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*. IEEE. 2006, 4–pp (cit. on p. 42).

[Ben+07] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. „Greedy layer-wise training of deep networks“. In: *Advances in neural information processing systems* 19 (2007), p. 153 (cit. on pp. 3, 11).

[Ben+09] Samy Bengio, Fernando Pereira, Yoram Singer, and Dennis Strelow. „Group sparse coding". In: *Advances in Neural Information Processing Systems*. 2009, pp. 82–89 (cit. on p. 25).

[BG09] Serhat S Bucak and Bilge Gunsel. „Incremental subspace learning via non-negative matrix factorization". In: *Pattern Recognition* 42.5 (2009), pp. 788–797 (cit. on pp. 4, 37, 44, 45, 49, 114).

[BH69] AE Bryson and Yu-Chi Ho. *Applied optimal control*. Vol. 8. 1969 (cit. on p. 10).

[Bis+95] Christopher M Bishop et al. „Neural networks for pattern recognition". In: (1995) (cit. on p. 14).

[BL+12] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. „Discriminative Non-negative Matrix Factorization for Multiple Pitch Estimation." In: *ISMIR*. Citeseer. 2012, pp. 205–210 (cit. on pp. 27, 37, 42, 47).

[BL07] James Bennett and Stan Lanning. „The netflix prize". In: *Proceedings of KDD cup and workshop*. Vol. 2007. 2007, p. 35 (cit. on pp. 4, 52, 53).

[Bla+02] John A Black, Antonio Paez, and Putu A Suthanaya. „Sustainable urban transportation: performance indicators and some analytical approaches". In: *Journal of urban planning and development* 128.4 (2002), pp. 184–209 (cit. on p. 100).

[BLC88] Sue Becker and Yann Le Cun. „Improving the convergence of back-propagation learning with second order methods". In: *Proceedings of the 1988 connectionist models summer school*. San Matteo, CA: Morgan Kaufmann. 1988, pp. 29–37 (cit. on pp. 9, 10).

[Ble+03] David M Blei, Andrew Y Ng, and Michael I Jordan. „Latent dirichlet allocation". In: *the Journal of machine Learning research* 3 (2003), pp. 993–1022 (cit. on pp. 30, 60, 61, 84).

[Bli+07] John Blitzer, Mark Dredze, and Fernando Pereira. „Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification". In: *ACL*. Vol. 7. Citeseer. 2007, pp. 440–447 (cit. on p. 59).

[BM01] Ella Bingham and Heikki Mannila. „Random projection in dimensionality reduction: applications to image and text data". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2001, pp. 245–250 (cit. on p. 36).

[Bot91] Léon Bottou. „UNE APPROCHE THEORIQUE DE L'APPRENTISSAGE CONNEXIONNISTE ET APPLICATIONS A LA RECONNAISSANCE DE LA PAROLE". PhD thesis. 1991 (cit. on pp. 3, 9).

[Bre+98] John S Breese, David Heckerman, and Carl Kadie. „Empirical analysis of predictive algorithms for collaborative filtering". In: *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 1998, pp. 43–52 (cit. on p. 52).

[Bro+06] Dirk Brockmann, Lars Hufnagel, and Theo Geisel. „The scaling laws of human travel". In: *Nature* 439.7075 (2006), pp. 462–465 (cit. on p. 100).

[Bur00] Robin Burke. „Knowledge-based recommender systems“. In: *Encyclopedia of library and information systems* 69.Supplement 32 (2000), pp. 175–186 (cit. on p. 52).

[Bur02] Robin Burke. „Hybrid recommender systems: Survey and experiments“. In: *UMUAI'02* 12.4 (2002), pp. 331–370 (cit. on p. 52).

[Bur98] Christopher JC Burges. „A tutorial on support vector machines for pattern recognition“. In: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167 (cit. on p. 88).

[Cai+08] Deng Cai, Xiaofei He, Xiaoyun Wu, and Jiawei Han. „Non-negative matrix factorization on manifold“. In: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE. 2008, pp. 63–72 (cit. on p. 46).

[Cai+09] Deng Cai, Xiaofei He, Xuanhui Wang, Hujun Bao, and Jiawei Han. „Locality Preserving Nonnegative Matrix Factorization.“ In: *IJCAI*. Vol. 9. 2009, pp. 1010–1015 (cit. on pp. 46, 47).

[Cai+11] Deng Cai, Xiaofei He, Jiawei Han, and Thomas S Huang. „Graph regularized nonnegative matrix factorization for data representation“. In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.8 (2011), pp. 1548–1560 (cit. on p. 46).

[Cea+12] Irina Ceapa, Chris Smith, and Licia Capra. „Avoiding the crowds: understanding tube station congestion patterns from trip data“. In: *ACM SIGKDD 2012*. ACM. 2012, pp. 134–141 (cit. on p. 100).

[Cha+06] Olivier Chapelle, Bernhard Schölkopf, Alexander Zien, et al. *Semi-supervised learning*. Vol. 2. MIT press Cambridge, 2006 (cit. on p. 47).

[Cha+10] Tanmoy Chakraborty, Eyal Even-Dar, Sudipto Guha, Yishay Mansour, and S Muthukrishnan. „Selective call out and real time bidding“. In: *Internet and Network Economics*. Springer, 2010, pp. 145–157 (cit. on p. 2).

[Cha+14] Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. „Return of the Devil in the Details: Delving Deep into Convolutional Nets“. In: *arXiv preprint arXiv:1405.3531* (2014) (cit. on pp. 3, 11, 121).

[Coa+11] Adam Coates, Andrew Y Ng, and Honglak Lee. „An analysis of single-layer networks in unsupervised feature learning“. In: *International Conference on Artificial Intelligence and Statistics*. 2011, pp. 215–223 (cit. on pp. 23, 26).

[Coa+13] Adam Coates, Brody Huval, Tao Wang, et al. „Deep learning with cots hpc systems“. In: *Proceedings of The 30th International Conference on Machine Learning*. 2013, pp. 1337–1345 (cit. on pp. 10, 11).

[Col+11] Ronan Collobert, Jason Weston, Léon Bottou, et al. „Natural language processing (almost) from scratch“. In: *The Journal of Machine Learning Research* 12 (2011), pp. 2493–2537 (cit. on pp. 3, 11, 29–31, 33, 121).

[Com94] Pierre Comon. „Independent component analysis, a new concept?“ In: *Signal processing* 36.3 (1994), pp. 287–314 (cit. on pp. 4, 37, 49).

[CPH05] Miguel A Carreira-Perpinan and Geoffrey E Hinton. „On contrastive divergence learning“. In: *Proceedings of the tenth international workshop on artificial intelligence and statistics*. Citeseer. 2005, pp. 33–40 (cit. on pp. 10, 12, 20).

[CS11] Youngmin Cho and Lawrence K Saul. „Nonnegative Matrix Factorization for Semi-supervised Dimensionality Reduction“. In: *arXiv preprint arXiv:1112.3714* (2011) (cit. on p. 48).

[Dea+12] Jeffrey Dean, Greg Corrado, Rajat Monga, et al. „Large scale distributed deep networks“. In: *Advances in Neural Information Processing Systems*. 2012, pp. 1223–1231 (cit. on pp. 10, 11, 18).

[DGX11] Mithun Das Gupta and Jing Xiao. „Non-negative matrix factorization as a feature selection tool for maximum margin classifiers“. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 2841–2848 (cit. on pp. 47, 48).

[Dia+10] Fernando Diaz, Donald Metzler, and Sihem Amer-Yahia. „Relevance and ranking in online dating systems“. In: *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2010, pp. 66–73 (cit. on pp. 2, 4, 51).

[Din+06] Chris Ding, Tao Li, and Michael Jordan. „Convex and semi-nonnegative matrix factorizations for clustering and low-dimension representation“. In: *Lawrence Berkeley National Laboratory, Tech. Rep. LBNL-60428* (2006) (cit. on p. 46).

[Don06] David L Donoho. „Compressed sensing“. In: *Information Theory, IEEE Transactions on* 52.4 (2006), pp. 1289–1306 (cit. on p. 25).

[Dor96] Georg Dorffner. „Neural networks for time series processing“. In: *Neural Network World*. Citeseer. 1996 (cit. on pp. 2, 29).

[Fan+08] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. „LIBLINEAR: A library for large linear classification“. In: *The Journal of Machine Learning Research* 9 (2008), pp. 1871–1874 (cit. on pp. 90, 91).

[FH10] Johannes Fürnkranz and Eyke Hüllermeier. *Preference learning*. Springer, 2010 (cit. on pp. 2, 4, 51).

[Foe+13] Stefan Foell, Gerd Kortuem, Reza Rawassizadeh, et al. „Mining temporal patterns of transport behaviour for predicting future transport usage“. In: *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*. ACM. 2013, pp. 1239–1248 (cit. on pp. 100, 109).

[Gan+09] Gayatree Ganu, Noemie Elhadad, and Amélie Marian. „Beyond the Stars: Improving Rating Predictions using Review Text Content.“ In: *WebDB*. 2009 (cit. on pp. 5, 30, 53, 59, 60, 71).

[Glo+11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. „Domain adaptation for large-scale sentiment classification: A deep learning approach“. In: *ICML'11*. 1. 2011, pp. 513–520 (cit. on pp. 11, 21, 22, 30, 33, 70, 71, 83, 84, 97).

[Gol02] John C Golias. „Analysis of traffic corridor impacts from the introduction of the new Athens Metro system“. In: *Journal of Transport Geography* 10.2 (2002), pp. 91–97 (cit. on p. 100).

[Gon+08] Marta C Gonzalez, Cesar A Hidalgo, and Albert-Laszlo Barabasi. „Understanding individual human mobility patterns“. In: *Nature* 453.7196 (2008), pp. 779–782 (cit. on p. 100).

[Gra12] Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Vol. 385. Springer, 2012 (cit. on pp. 2, 3, 11, 28, 29, 114, 121).

[GVL12] Gene H Golub and Charles F Van Loan. *Matrix computations*. Vol. 3. JHU Press, 2012 (cit. on pp. 3, 4, 36, 37, 49).

[GZ05] Edward F Gonzalez and Yin Zhang. „Accelerating the Lee-Seung algorithm for non-negative matrix factorization". In: *Dept. Comput. & Appl. Math., Rice Univ., Houston, TX, Tech. Rep. TR-05-02* (2005) (cit. on p. 109).

[Her10] Ryan Jay Herring. „Real-time traffic modeling and estimation with streaming probe data using machine learning". In: (2010) (cit. on p. 100).

[Hin+12] Geoffrey Hinton, Li Deng, Dong Yu, et al. „Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *Signal Processing Magazine, IEEE* 29.6 (2012), pp. 82–97 (cit. on p. 11).

[HL04] Minqing Hu and Bing Liu. „Mining and summarizing customer reviews". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, pp. 168–177 (cit. on p. 59).

[HL06] Minqing Hu and Bing Liu. „Opinion extraction and summarization on the web". In: *AAAI*. Vol. 7. 2006, pp. 1621–1624 (cit. on pp. 58, 59).

[Hof+99] Thomas Hofmann, Jan Puzicha, and Michael I Jordan. „Learning from dyadic data". In: *Advances in neural information processing systems* (1999), pp. 466–472 (cit. on p. 36).

[Hor+12] Inbal Horev, Ori Bryt, and Ron Rubinstein. „Adaptive image compression using sparse dictionaries". In: *Systems, Signals and Image Processing (IWSSIP), 2012 19th International Conference on*. IEEE. 2012, pp. 592–595 (cit. on p. 35).

[Hot33] Harold Hotelling. „Analysis of a complex of statistical variables into principal components." In: *Journal of educational psychology* 24.6 (1933), p. 417 (cit. on pp. 4, 35–37, 49).

[Hoy02] Patrik O Hoyer. „Non-negative sparse coding". In: *Neural Networks for Signal Processing, 2002. Proceedings of the 2002 12th IEEE Workshop on*. IEEE. 2002, pp. 557–565 (cit. on pp. 4, 25, 37, 45, 49, 107–109).

[Hoy04] Patrik O Hoyer. „Non-negative matrix factorization with sparseness constraints". In: *The Journal of Machine Learning Research* 5 (2004), pp. 1457–1469 (cit. on pp. 4, 25, 37, 45, 121).

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. „Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507 (cit. on pp. 3, 10, 11, 20–22, 33).

[Jam+10] Aruna Jammalamadaka, Swapna Joshi, Shanmugavadivel Karthikeyan, and BS Manjunath. „Discriminative Basis Selection using Non-negative Matrix Factorization". In: *Pattern Recognition (ICPR), 2010 20th International Conference on*. IEEE. 2010, pp. 1533–1536 (cit. on p. 47).

[JL08] Nitin Jindal and Bing Liu. „Opinion spam and analysis". In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM. 2008, pp. 219–230 (cit. on pp. 1, 63, 93).

[Kan+11] Paul B Kantor, Lior Rokach, Francesco Ricci, and Bracha Shapira. *Recommender systems handbook*. Springer, 2011 (cit. on p. 2).

[KB11] Yehuda Koren and Robert Bell. „Advances in collaborative filtering". In: *Recommender Systems Handbook*. Springer, 2011, pp. 145–186 (cit. on pp. 4, 36, 53).

[Koh82] Teuvo Kohonen. „Self-organized formation of topologically correct feature maps". In: *Biological cybernetics* 43.1 (1982), pp. 59–69 (cit. on p. 20).

[Kor+09] Yehuda Koren, Robert Bell, and Chris Volinsky. „Matrix factorization techniques for recommender systems". In: *Computer* 42.8 (2009), pp. 30–37 (cit. on pp. 5, 44, 48, 53, 56).

[Kor08] Yehuda Koren. „Factorization meets the neighborhood: a multifaceted collaborative filtering model". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 426–434 (cit. on pp. 5, 52, 53).

[Kot+07] Irene Kotsia, Stefanos Zafeiriou, and Ioannis Pitas. „A Novel Discriminant Non-Negative Matrix Factorization Algorithm With Applications to Facial Image Characterization Problems." In: *IEEE Transactions on Information Forensics and Security* 2.3-2 (2007), pp. 588–595 (cit. on p. 47).

[KR09] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. Vol. 344. John Wiley & Sons, 2009 (cit. on p. 35).

[Kri+12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cit. on p. 2).

[Law+97] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. „Face recognition: A convolutional neural-network approach". In: *Neural Networks, IEEE Transactions on* 8.1 (1997), pp. 98–113 (cit. on p. 1).

[LC+90] Y Le Cun, B Boser, JS Denker, et al. „Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems*. Citeseer. 1990 (cit. on pp. 1, 3, 9, 10, 18, 33).

[LCB04] Yann Le Cun and Leon Bottou. „Large scale online learning". In: *Advances in neural information processing systems* 16 (2004), p. 217 (cit. on pp. 16, 89).

[LD06] Tao Li and Chris Ding. „The relationships among various nonnegative matrix factorization methods for clustering". In: *Data Mining, 2006. ICDM'06. Sixth International Conference on*. IEEE. 2006, pp. 362–371 (cit. on p. 46).

[Le+13] QV Le, MA Ranzato, R Monga, et al. „Building high-level features using large scale unsupervised learning". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 8595–8598 (cit. on p. 22).

[Lee+06] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. „Efficient sparse coding algorithms". In: *Advances in neural information processing systems*. 2006, pp. 801–808 (cit. on pp. 25, 43).

[Lee+08] Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. „Sparse deep belief net model for visual area V2“. In: *Advances in neural information processing systems*. 2008, pp. 873–880 (cit. on p. 25).

[Li+09] Tao Li, Yi Zhang, and Vikas Sindhwani. „A non-negative matrix tri-factorization approach to sentiment classification with lexical prior knowledge“. In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1*. Association for Computational Linguistics. 2009, pp. 244–252 (cit. on pp. 46, 48).

[Lia+10] Zhizheng Liang, Youfu Li, and Tuo Zhao. „Projected gradient method for kernel discriminant nonnegative matrix factorization and the applications“. In: *Signal Processing* 90.7 (2010), pp. 2150–2163 (cit. on p. 47).

[Lie+12] Thomas Liebig, Zhao Xu, Michael May, and Stefan Wrobel. „Pedestrian quantity estimation with trajectory patterns“. In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2012, pp. 629–643 (cit. on p. 100).

[Lin04] Chin-Yew Lin. „Rouge: A package for automatic evaluation of summaries“. In: *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*. 2004, pp. 74–81 (cit. on pp. 54, 58, 69, 72).

[Lin07] Chih-Jen Lin. „Projected gradient methods for nonnegative matrix factorization“. In: *Neural computation* 19.10 (2007), pp. 2756–2779 (cit. on pp. 43, 109).

[Liu+06] Yi Liu, Rong Jin, and Liu Yang. „Semi-supervised multi-label learning by constrained non-negative matrix factorization“. In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 21. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2006, p. 421 (cit. on p. 47).

[LM14] Quoc V Le and Tomas Mikolov. „Distributed Representations of Sentences and Documents“. In: *arXiv preprint arXiv:1405.4053* (2014) (cit. on pp. 3, 11, 30, 32, 33, 83, 121).

[Lou+14] Thomas Louail, Maxime Lenormand, Oliva García Cantú, et al. „From mobile phone data to the spatial structure of cities“. In: *arXiv preprint arXiv:1401.4540* (2014) (cit. on p. 100).

[Low+10] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, et al. „Graphlab: A new framework for parallel machine learning“. In: *arXiv preprint arXiv:1006.4990* (2010) (cit. on p. 92).

[LS01] Daniel D Lee and H Sebastian Seung. „Algorithms for non-negative matrix factorization“. In: *Advances in neural information processing systems*. 2001, pp. 556–562 (cit. on pp. 3, 4, 37, 42, 43, 45, 49, 109, 121).

[LS99] Daniel D Lee and H Sebastian Seung. „Learning the parts of objects by non-negative matrix factorization“. In: *Nature* 401.6755 (1999), pp. 788–791 (cit. on pp. 4, 37, 42).

[LW10] Haifeng Liu and Zhaohui Wu. „Non-negative matrix factorization with constraints“. In: *Twenty-Fourth AAAI Conference on Artificial Intelligence*. 2010 (cit. on p. 47).

[Mai+08] Julien Mairal, Francis Bach, Jean Ponce, Guillermo Sapiro, and Andrew Zisserman. „Discriminative learned dictionaries for local image analysis“. In: *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE. 2008, pp. 1–8 (cit. on pp. 37, 47, 115).

[Mai+10] Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. „Online learning for matrix factorization and sparse coding“. In: *The Journal of Machine Learning Research* 11 (2010), pp. 19–60 (cit. on pp. 3, 25, 37, 44, 45, 107).

[McA+12] Julian McAuley, Jure Leskovec, and Dan Jurafsky. „Learning attitudes and attributes from multi-aspect reviews“. In: *Data Mining (ICDM), 2012 IEEE 12th International Conference on*. IEEE. 2012, pp. 1020–1025 (cit. on p. 63).

[MH04] Matthew R McLaughlin and Jonathan L Herlocker. „A collaborative filtering algorithm and evaluation metric that accurately model the user experience“. In: *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2004, pp. 329–336 (cit. on p. 52).

[Mik+13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. „Distributed representations of words and phrases and their compositionality“. In: *Advances in Neural Information Processing Systems*. 2013, pp. 3111–3119 (cit. on pp. 1, 3, 11, 32, 83, 121).

[Mit+10] Kaushik Mitra, Sameer Sheorey, and Rama Chellappa. „Large-scale matrix factorization with missing data under additional constraints“. In: *Advances in Neural Information Processing Systems*. 2010, pp. 1651–1659 (cit. on p. 45).

[ML10] Brian McFee and Gert R Lanckriet. „Metric learning to rank“. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010, pp. 775–782 (cit. on pp. 4, 51).

[ML13a] Julian McAuley and Jure Leskovec. „Hidden factors and hidden topics: understanding rating dimensions with review text“. In: *Proceedings of the 7th ACM conference on Recommender systems*. ACM. 2013, pp. 165–172 (cit. on pp. 5, 53, 59, 60, 64, 71, 93).

[ML13b] Julian John McAuley and Jure Leskovec. „From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews“. In: *Proceedings of the 22nd international conference on World Wide Web*. International World Wide Web Conferences Steering Committee. 2013, pp. 897–908 (cit. on pp. 29, 59, 76, 93).

[Mon+12] Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller, eds. *Neural Networks: Tricks of the Trade, Reloaded*. 2nd edn. Vol. 7700. Lecture Notes in Computer Science (LNCS). Springer, 2012 (cit. on pp. 10, 16, 19, 24, 33, 57).

[Ng04] Andrew Y Ng. „Feature selection, L 1 vs. L 2 regularization, and rotational invariance“. In: *Proceedings of the twenty-first international conference on Machine learning*. ACM. 2004, p. 78 (cit. on p. 24).

[Pan+02] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. „Thumbs up?: sentiment classification using machine learning techniques“. In: *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*. Association for Computational Linguistics. 2002, pp. 79–86 (cit. on pp. 5, 87).

[Pap+02] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. „BLEU: a method for automatic evaluation of machine translation“. In: *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics. 2002, pp. 311–318 (cit. on p. 76).

[Par08] Sun Park. „Personalized summarization agent using non-negative matrix factorization“. In: *PRICAI 2008: Trends in Artificial Intelligence*. Springer, 2008, pp. 1034–1038 (cit. on p. 42).

[PB07] Michael J Pazzani and Daniel Billsus. „Content-based recommendation systems“. In: *The adaptive web*. Springer, 2007, pp. 325–341 (cit. on p. 52).

[PC14] Pedro Pinheiro and Ronan Collobert. „Recurrent convolutional neural networks for scene labeling“. In: *Proceedings of The 31st International Conference on Machine Learning*. 2014, pp. 82–90 (cit. on p. 29).

[Pea01] Karl Pearson. „On lines and planes of closest fit to systems of points in space“. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2.11 (1901), pp. 559–572 (cit. on pp. 4, 35–37).

[Pen+12] Chengbin Peng, Xiaogang Jin, Ka-Chun Wong, Meixia Shi, and Pietro Liò. „Collective human mobility pattern from taxi trips in urban area“. In: *PloS one* 7.4 (2012), e34487 (cit. on pp. 100, 101).

[PL08] Bo Pang and Lillian Lee. „Opinion mining and sentiment analysis“. In: *Foundations and trends in information retrieval* 2.1-2 (2008), pp. 1–135 (cit. on pp. 2, 62, 81, 82, 84, 87, 90, 91, 97).

[PN07] Xuan-Hieu Phan and Cam-Tu Nguyen. *GibbsLDA++: A C/C++ implementation of latent Dirichlet allocation (LDA)*. 2007 (cit. on pp. 61, 63).

[Pot11] Vamsi K Potluru. „Understanding and Exploiting the Connections between NMF and SVM.“ In: *ICDM Workshops*. 2011, pp. 1207–1210 (cit. on p. 47).

[Pou+] Mickaël Poussevin, Nicolas Baskiotis, Vincent Guigue, and Patrick Gallinari. „Mining ticketing logs for usage characterization with nonnegative matrix factorization“. In: *SenseML 2014–ECML Workshop* (cit. on p. 114).

[Pou+14a] Mickaël Poussevin, Vincent Guigue, and Patrick Gallinari. „Extended Recommendation Framework: Generating the Text of a User Review as a Personalized Summary“. In: *arXiv preprint arXiv:1412.5448* (2014) (cit. on pp. 76, 97).

[Pou+14b] Mickaël Poussevin, Elie Guardia-Sebaoun, Vincent Guigue, Patrick Gallinari, et al. „Recommandation par combinaison de filtrage collaboratif et d’analyse de sentiments“. In: *Actes de la onzi eme COnf erence en Recherche d’Information et Applications*. 2014 (cit. on pp. 76, 97).

[PZ11] Ji-Yuan Pan and Jiang-She Zhang. „Large margin based nonnegative matrix factorization and partial least squares regression for face recognition“. In: *Pattern Recognition Letters* 32.14 (2011), pp. 1822–1835 (cit. on pp. 36, 47).

[Raf+12] Abdelhalim Rafrafi, Vincent Guigue, and Patrick Gallinari. „Coping with the Document Frequency Bias in Sentiment Classification.“ In: *ICWSM*. 2012 (cit. on pp. 24, 25, 89).

[Ran+07] M Ranzato, Fu Jie Huang, Y-L Boureau, and Yann LeCun. „Unsupervised learning of invariant feature hierarchies with applications to object recognition“. In: *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*. IEEE. 2007, pp. 1–8 (cit. on p. 26).

[Ran+13] Andry Randriamanamihaga, Etienne Côme, Latifa Oukhellou, and Gérard Govaert. „Clustering the Vélib origin-destinations flows by means of Poisson mixture models“. In: *ESANN 2013*. 2013 (cit. on p. 100).

[Res+94] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. „GroupLens: an open architecture for collaborative filtering of netnews“. In: *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM. 1994, pp. 175–186 (cit. on pp. 52, 53).

[Rif+11a] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. „Contractive auto-encoders: Explicit invariance during feature extraction“. In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 833–840 (cit. on p. 26).

[Rif+11b] Salah Rifai, Grégoire Mesnil, Pascal Vincent, et al. „Higher order contractive auto-encoder“. In: *Proceedings of the 2011 European conference on Machine learning and knowledge discovery in databases-Volume Part II*. Springer-Verlag. 2011, pp. 645–660 (cit. on p. 26).

[RL13] Jason Tyler Rolfe and Yann LeCun. „Discriminative recurrent sparse auto-encoders“. In: *arXiv preprint arXiv:1301.3775* (2013) (cit. on p. 27).

[RM98] Russell D. Reed and Robert J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Cambridge, MA, USA: MIT Press, 1998 (cit. on p. 20).

[Ros58] Frank Rosenblatt. „The perceptron: a probabilistic model for information storage and organization in the brain.“ In: *Psychological review* 65.6 (1958), p. 386 (cit. on pp. 9, 13, 33).

[RS08] Marc'Aurelio Ranzato and Martin Szummer. „Semi-supervised learning of compact document representations with deep networks“. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 792–799 (cit. on pp. 11, 27).

[Rum+88] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. „Learning representations by back-propagating errors“. In: *Cognitive modeling* (1988) (cit. on pp. 9, 10, 19).

[Rus+14] Olga Russakovsky, Jia Deng, Hao Su, et al. *ImageNet Large Scale Visual Recognition Challenge*. 2014. eprint: `arXiv:1409.0575` (cit. on p. 1).

[Sch+97] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. „Kernel principal component analysis“. In: *Artificial Neural Networks—ICANN'97*. Springer, 1997, pp. 583–588 (cit. on p. 37).

[Sch+99] J Ben Schafer, Joseph Konstan, and John Riedl. „Recommender systems in e-commerce“. In: *Proceedings of the 1st ACM conference on Electronic commerce*. ACM. 1999, pp. 158–166 (cit. on p. 51).

[SG08] Ajit P Singh and Geoffrey J Gordon. „Relational learning via collective matrix factorization“. In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2008, pp. 650–658 (cit. on pp. 44, 45).

[Soc+11] Richard Socher, Jeffrey Pennington, Eric H Huang, Andrew Y Ng, and Christopher D Manning. „Semi-supervised recursive autoencoders for predicting sentiment distributions“. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2011, pp. 151–161 (cit. on p. 30).

[Son+10] Chaoming Song, Zehui Qu, Nicholas Blumm, and Albert-László Barabási. „Limits of predictability in human mobility“. In: *Science* 327.5968 (2010), pp. 1018–1021 (cit. on p. 100).

[Sou+87] F Fogelman Soulie, P Gallinari, and S Thiria. „Learning and associative memory“. In: *Pattern Recognition Theory and Applications*. Springer, 1987, pp. 249–268 (cit. on pp. 3, 9, 10).

[SS02] Bernhard Schölkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2002 (cit. on pp. 88, 89).

[Sze+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, et al. „Intriguing properties of neural networks“. In: *arXiv preprint arXiv:1312.6199* (2013) (cit. on pp. 3, 11, 18, 23, 121).

[TA77] Andrej Nikolaevich Tikhonov and Vasiliy Yakovlevich Arsenin. „Solutions of ill-posed problems“. In: (1977) (cit. on p. 24).

[Tan+11] Chenhao Tan, Lillian Lee, Jie Tang, et al. „User-level sentiment analysis incorporating social networks“. In: *KDD'11*. ACM. 2011, pp. 1397–1405 (cit. on p. 59).

[Tan+12] Chenhao Tan, Evgeniy Gabrilovich, and Bo Pang. „To each his own: personalized content selection based on text comprehensibility“. In: *ICWDM'12*. ACM. 2012, pp. 233–242 (cit. on p. 59).

[TB99] Michael E Tipping and Christopher M Bishop. „Probabilistic principal component analysis“. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61.3 (1999), pp. 611–622 (cit. on p. 48).

[Tee+05] Jaime Teevan, Susan T Dumais, and Eric Horvitz. „Personalizing search via automated analysis of interests and activities“. In: *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2005, pp. 449–456 (cit. on pp. 2, 4, 51).

[Tho+02] Christopher G Thomas, Richard A Harshman, and Ravi S Menon. „Noise reduction in BOLD-based fMRI using component analysis“. In: *Neuroimage* 17.3 (2002), pp. 1521–1537 (cit. on p. 37).

[Tib96] Robert Tibshirani. „Regression shrinkage and selection via the lasso“. In: *Journal of the Royal Statistical Society. Series B (Methodological)* (1996), pp. 267–288 (cit. on pp. 24, 44).

[VDW+11] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. „The NumPy array: a structure for efficient numerical computation“. In: *Computing in Science & Engineering* 13.2 (2011), pp. 22–30 (cit. on p. 109).

[Vin+08] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. „Extracting and composing robust features with denoising autoencoders“. In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 1096–1103 (cit. on pp. 11, 33).

[Wan+09] Changhu Wang, Shuicheng Yan, Lei Zhang, and Hongjiang Zhang. „Non-negative semi-supervised learning“. In: *International Conference on Artificial Intelligence and Statistics*. 2009, pp. 575–582 (cit. on pp. 37, 47).

[Wan+11] Dashun Wang, Dino Pedreschi, Chaoming Song, Fosca Giannotti, and Albert-László Barabási. „Human mobility, social ties, and link prediction“. In: *ACM SIGKDD 2011*. ACM. 2011, pp. 1100–1108 (cit. on p. 100).

[Wan+12] Quan Wang, Zheng Cao, Jun Xu, and Hang Li. „Group matrix factorization for scalable topic modeling“. In: *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2012, pp. 375–384 (cit. on p. 47).

[Wan+13] Xinggang Wang, Baoyuan Wang, Xiang Bai, Wenyu Liu, and Zhuowen Tu. „Max-margin multiple-instance dictionary learning“. In: *Proceedings of The 30th International Conference on Machine Learning*. 2013, pp. 846–854 (cit. on p. 47).

[Wer90] Paul J Werbos. „Backpropagation through time: what it does and how to do it“. In: *Proceedings of the IEEE* 78.10 (1990), pp. 1550–1560 (cit. on p. 28).

[Xu+03] 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*. ACM. 2003, pp. 267–273 (cit. on p. 42).

[Xu+13] Minjie Xu, Jun Zhu, and Bo Zhang. „Fast max-margin matrix factorization with data augmentation“. In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013, pp. 978–986 (cit. on p. 48).

[Yin+14] Jiangtao Yin, Lixin Gao, and Zhongfei Mark Zhang. „Scalable Nonnegative Matrix Factorization with Block-wise Updates“. In: *Machine Learning and Knowledge Discovery in Databases*. Springer, 2014, pp. 337–352 (cit. on pp. 4, 38, 45, 49, 57).

[YO10] Zhirong Yang and Erkki Oja. „Linear and nonlinear projective nonnegative matrix factorization“. In: *Neural Networks, IEEE Transactions on* 21.5 (2010), pp. 734–749 (cit. on p. 37).

[Zaf+06] Stefanos Zafeiriou, Anastasios Tefas, Ioan Buciu, and Ioannis Pitas. „Exploiting discriminant information in nonnegative matrix factorization with application to frontal face verification“. In: *Neural Networks, IEEE Transactions on* 17.3 (2006), pp. 683–695 (cit. on p. 42).

[ZH05] Hui Zou and Trevor Hastie. „Regularization and variable selection via the elastic net“. In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67.2 (2005), pp. 301–320 (cit. on p. 24).

[Zha+14] Yongfeng Zhang, Guokun Lai, Min Zhang, et al. „Explicit Factor Models for Explainable Recommendation based on Phrase-level Sentiment Analysis“. In: (2014) (cit. on p. 59).

List of Figures

List of Tables