ALMA MATER STUDIORUM — UNIVERSITÀ DI BOLOGNA

DISI - Dipartimento di Informatica: Scienza e Ingegneria
PhD in Computer Science and Engineering

Ciclo XXXI
Settore Concorsuale: 09/H1
Settore Disciplinare: ING-INF/05

BIG DATA MINING AND MACHINE LEARNING TECHNIQUES APPLIED TO REAL WORLD SCENARIOS

Candidato

Dott. Ing. ANDREA PAGLIARANI

Supervisore

Chiar.mo Prof. Ing. GIANLUCA MORO

Tutor

Chiar.mo Prof. Ing. CLAUDIO SARTORI

Coordinatore

Chiar.mo Prof. Ing. PAOLO CIACCIA

FINAL EXAMINATION YEAR 2019

Be modest and determined,
and do what you love to do,
always.

Acknowledgements

Most of the material that got integrated as a single work in this thesis has been written in publications I co-authored. I am thankful to those great researchers I have collaborated with, for their guidance, their suggestions and their help. I would like to thank first Gianluca Moro, who carefully supervised me in the past three years and whose deep expertise has been pivotal for my research. Claudio Sartori, who co-supervised me, sharing lots of essential research ideas, also deserves a special mention. I am grateful for the opportunity Claudio gave me to contribute to TOREADOR, an European project that kickstarted me in the world of research. I also desire to thank Giacomo Domeniconi and Roberto Pasolini, who shared the APICe laboratory with me and tried to improve my technical skills. A special thank to Elizabeth Daly, who supervised me during my three-month research period at IBM Research - Ireland Lab. Finally, I want to thank these other great researchers and professionals I have collaborated with: Oznur Alkan, Adi Botea, Stefano Lodi, Beniamino Di Martino, Cesare Bandirali, Salvatore D'Angelo, Antonio Esposito, Karin Pasini.

Andrea Pagliarani, 8th February 2019

Contents

Abstract

Data mining techniques allow the extraction of valuable information from heterogeneous and possibly very large data sources, which can be either structured or unstructured. Unstructured data, such as text files, social media, mobile data, are much more than structured data, and grow at a higher rate. Their high volume and the inherent ambiguity of natural language make unstructured data very hard to process and analyze. Appropriate text representations are therefore required in order to capture word semantics as well as to preserve statistical information, e.g. word counts. In Big Data scenarios, scalability is also a primary requirement. Data mining and machine learning approaches should take advantage of large-scale data, exploiting abundant information and avoiding the curse of dimensionality. The goal of this thesis is to enhance text understanding in the analysis of big data sets, introducing novel techniques that can be employed for the solution of real world problems. The presented Markov methods temporarily achieved the state-of-the-art on well-known Amazon reviews corpora for cross-domain sentiment analysis, before being outperformed by deep approaches in the analysis of large data sets. A noise detection method for the identification of relevant tweets leads to 88.9% accuracy in the Dow Jones Industrial Average daily prediction, which is the best result in literature based on social networks. Dimensionality reduction approaches are used in combination with LinkedIn users' skills to perform job recommendation. A framework based on deep learning and Markov Decision Process is designed with the purpose of modeling job transitions and recommending pathways towards a given career goal. Finally, parallel primitives for vendor-agnostic implementation of Big Data mining algorithms are introduced to foster multi-platform deployment, code reuse and optimization.

1 About this thesis

This chapter briefly introduces the content of this work, the main problems tackled, why they are relevant, and how the thesis is organized. Also, a list of publications that directly contribute to this work is provided.

This work contributes to the fields of Big Data mining and machine learning, which aim at discovering interesting, unexpected or valuable structures in large datasets. Recently data have grown at an unprecedented scale, due to the fast development of networking, data storage, data collection capacity, and to the spread of data sources different from traditional databases. Such data sources, namely social media, sensors, mobile data, weblogs, and so on, essentially are sources of unstructured data, mainly expressed in text format. The result of the growth of unstructured information are datasets that cannot easily be managed with current methodologies and software tools due to their large size and complexity. Such datasets represent a great opportunity to enhance text understanding, but require approaches in order to be processed and analyzed appropriately. This thesis introduces novel techniques to mine semantics from text in order to improve data mining and machine learning algorithms, focusing on hot research threads, such as:

Sentiment analysis Computational treatment of opinion, appraisals, attitudes, and emotions towards entities, individuals, issues, events, topics and their attributes. Traditional approaches usually require a different set of labeled data anytime a new domain has to be analyzed. Moreover, most algorithms are affected by the curse of dimensionality, which hinders scalability and prevents from making use of the whole available information. A possible improvement consists in designing innovative techniques for domain adaptation, so as to pave the way for model reuse and to deal with the scarcity of labeled data, needed to learn accurate models. Moreover, this thesis relies on existing work and carries out an analysis on which characteristics of deep learning architectures contribute the most to domain adaptation.

Stock market analysis Evaluation of a particular trading instrument, an investment sector, or the market as a whole. Stock market analysis is a primary interest for finance and traders. Differently from the classical trend analysis, this work exploits data from social media to detect and predict stock market variations.

Recommender systems for job search Finding the right job is one of the most relevant social problems, since everyone has to look for a job at least once, but possibly multiple times in life. Job searching is a time consuming activity, and the choice of the right job is awkward, because job satisfaction considerably affects life condition. This thesis first starts an investigation on the use of social networks to match candidates with jobs. Then it paves the way for an emerging research thread, introducing a framework to perform career pathway recommendation, namely suggesting a sequence of jobs to the user so as to lead her to a predefined career goal.

Big Data Analytics In the age of Big Data, scalable algorithm implementations as well as powerful computational resources are required. Big Data platforms and cloud platforms support programmers in coding algorithms, and offer optimized infrastructures for their execution. Code portability among platforms remains an open issue due to cost of service, technical and managerial factors. Other than discussing the problem, this thesis exploits existing work in order to propose parallel programming primitives to implement Big Data mining algorithms in an agnostic fashion.

1.1 Structure of this thesis

The thesis is organized in six parts.

Part I introduces the work, describing the background and motivating the thesis. The age of Big Data is presented, including challenges, opportunities and open issues. The focus is mainly on the research threads this thesis contributes to, namely sentiment analysis, stock market analysis, recommender systems for job search, and Big Data Analytics.

Part II describes novel algorithms and methods for sentiment analysis, pointing out their practical utilities in real world Big Data scenarios. The first contribution of this thesis is introduced, namely a Markov chain method that performs both transfer learning and sentiment classification. Also, the characteristics of deep architectures that support domain adaptation are investigated and discussed.

Part III describes some methods based on social media mining for stock market analysis. A Twitter-based approach exploiting noise detection techniques is introduced for the prediction of Dow Jones Industrial Average variations. Moreover, based on such a prediction, a trading protocol performing buy/sell operations is proposed to support investors.

Part IV describes new methods for job search that help people finding and getting the desired job. A job recommendation approach based on LinkedIn users' profiles is introduced to match candidates with jobs. Also, a novel framework is proposed to recommend a pathway to users towards a well-defined, possibly long term, career goal.

Part V introduces parallel programming primitives for vendor-agnostic implementation of Big Data mining algorithms. Some notable data mining algorithms are implemented using such parallel primitives. The transformation process from a vendor-agnostic implementation to a vendor-specific platform is also shown.

Part VI draws conclusions and paves the way for future work.

1.2 List of publications

The list of publications that directly contribute to this thesis is shown below. Note that the authors appear in alphabetical order in the papers listed below, apart from the works titled "Transfer learning in sentiment classification with deep neural networks", "Prediction and trading of Dow Jones from Twitter: a boosting text mining method with relevant tweets identification", and "A skillset-based approach to modelling job transitions and recommending career pathways", where the authors are listed in descending order of contribution.

- Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. Markov chain based method for in-domain and cross-domain sentiment classification. In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR, (IC3K 2015)*, pages 127–137. INSTICC, SciTePress, 2015

- Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. Cross-domain sentiment classification via polarity-driven state transitions in a markov model. In *Fred A., Dietz J., Aveiro D., Liu K., Filipe J. (eds) Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2015. Communications in Computer and Information Science, vol 631. Springer, Cham*

- Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, Karin Pasini, and Roberto Pasolini. Job recommendation from semantic similarity of linkedin users' skills. In *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM,*, pages 270–277. INSTICC, SciTePress, 2016

- Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. On deep learning in cross-domain sentiment classification. In *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR,*, pages 50–60. INSTICC, SciTePress, 2017

- Andrea Pagliarani, Gianluca Moro, Roberto Pasolini, and Giacomo Domeniconi. Transfer learning in sentiment classification with deep neural networks. In *Fred A., Dietz J., Aveiro D., Liu K., Filipe J. (eds) Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2017. Communications in Computer and Information Science*. To appear

- Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. Learning to predict the stock market dow jones index detecting and mining relevant tweets. In *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR,*, pages 165–172. INSTICC, SciTePress, 2017

- Gianluca Moro, Roberto Pasolini, Giacomo Domeniconi, Andrea Pagliarani, and Andrea Roli. Prediction and trading of dow jones from twitter: A boosting text mining method with relevant tweets identification. In *Fred A., Dietz J., Aveiro D., Liu K., Filipe J. (eds) Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2017. Communications in Computer and Information Science*. To appear

- Gianluca Moro, Andrea Pagliarani, Roberto Pasolini, and Claudio Sartori. Cross-domain & in-domain sentiment analysis with memory-based deep neural networks. In *10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR,*, 2018. To appear

- Cesare Bandirali, Stefano Lodi, Gianluca Moro, Andrea Pagliarani, and Claudio Sartori. Parallel primitives for vendor-agnostic implementation of big data mining algorithms. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 396–401, May 2018

- Andrea Pagliarani, Oznur Alkan, Adi Botea, Gianluca Moro, and Elizabeth Daly. A skillset-based approach to modelling job transitions and recommending career pathways. To be submitted to journal Expert Systems with Applications

Part I

Background and Motivation

2
The age of big data

This chapter introduces the age of Big Data, providing the reader with a background of such a historical and technological phase.

The reasons that brought to the widespread diffusion of data are discussed, with particular emphasis on technological aspects. Beyond outlining the main characteristics of big data, the challenges as well as the opportunities that such massive corpora offer are pointed out. Also, the technologies that facilitate the growth of data are summarized, identifying their salient features and relationships with Big Data.

After this general overview, the rest of the thesis will focus on data mining and machine learning methods, with particular emphasis on some hot research threads, such as sentiment analysis, stock market analysis, recommender systems for job search, and big data analytics.

2.1 What is big data?

Heterogeneous devices are pervasively present around us, such as Radio-Frequency IDentification (RFID), tags, sensors, actuators, mobile phones, etc. These objects are equipped with some kind of network connections, are able to interact with each other and cooperate with their neighbors to reach common goals [AIM10]. This scenario, known as the Internet of Things (IoT), has recently brought to a dramatic increase in the amount of data. Apart from IoT data, servers are also filled with Web data, which arise increasingly faster from social networks, blogs, forums, and so on.

When the Gutenberg press was invented, there was a doubling of information stock every 50 years. In 2011, a report from International Data Corporation (IDC) stated that the overall created and copied data volume in the world was 1.8ZB ($\approx 10^{21}$ B) [GR11]. As of 2012, data volume grows by 2.5 exabytes per day, and it is more than doubling every two or three years [Loh12, MBD+12]. Figure 2.1 illustrates the rapid global growth of data volume.

Big data have an enormous potential value for both research and business. Two premier scientific journals, Nature and Science, opened special columns to discuss the challenges and impacts of big data [Nat08, Sci11]. Also, issues on big data are covered in public media, such as New York Times [Loh12], National Public Radio [Nog11a, Nog11b], and The Economist [Cuk10, blo11]. Information extraction from big data may support the decision process of a

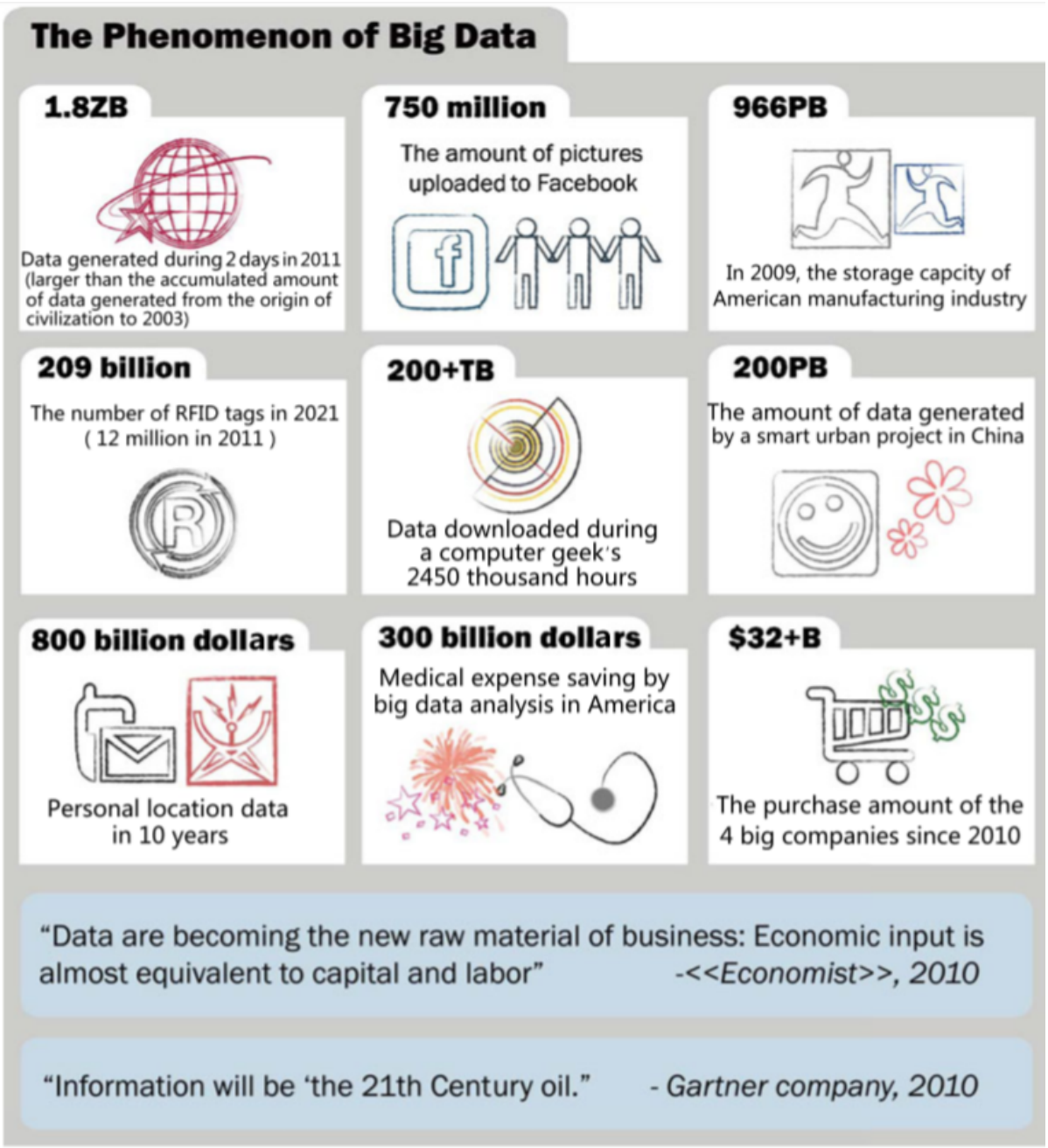


Figure 2.1: The continuously increasing big data. *Source: [CML14].*

large variety of organizations, from public and private enterprises to the financial sector, political parties, etc., enabling managers to make more conscious choices.

The increasing amount of data, along with extraordinary opportunities, brings about many challenging problems that require quick solutions. Data generation is fostered by the advances of information technology (IT), which make really fast to produce and store data. For example, on average, 72 hours of videos are uploaded to YouTube in every minute[JW14]. Data sources are typically heterogenous and widely distributed, so that the rapid growth of massive data makes it hard their gathering and integration.

The advent of cloud computing and the Internet of Things further promote the sharp growth of data. Cloud computing provides on demand network access to a shared pool of computing resources (e.g. networks, servers, storage, applications, and services) that can rapidly be provisioned and released with minimal management effort or service provider interaction. Resource usage is automatically controlled and optimized, and can be monitored for both the provider and the consumer of the utilized service in order to provide transparency [MG+11]. In the IoT paradigm, sensors all over the world are collecting and transmitting data to be stored and processed in the cloud. In terms of quantity, such data and their mutual relations are much more than the IT architectures and infrastructures of existing enterprises can actually deal with. Consequently, storing and managing such massive heterogeneous datasets with moderate hardware and software requirements become a challenge.

In order to disclose interesting properties and effectively support decision making, a dataset has to be mined at different levels during the analysis, modeling, visualization, and forecasting. Indeed, the heterogeneity, scalability, real-time, complexity, and privacy of big data pose different challenges that cannot trivially be solved, and that often require multiple approaches to be adequately managed.

2.1.1 The features of big data

Although people have different opinions on the correct definition of Big data, their importance has been generally recognized. The various definitions clarify the profound social, economic and technological connotations of big data. The common conception is that big data involve datasets that cannot be managed, processed and analyzed by traditional hardware and software tools within a tolerable time.

The very first definition is found in a research report by Laney in 2001, who defined challenges and opportunities brought about by increased data with a 3Vs model that considered Volume, Velocity, and Variety [Lan01].

Volume Data scale becomes increasingly big. Large-scale datasets are much more difficult to process and analyze through traditional softwares and mining algorithms.

Velocity Data grow very fast, typically real-time, especially if we think about sensors data or social media content, just to mention some. Therefore, data collection and analysis must be

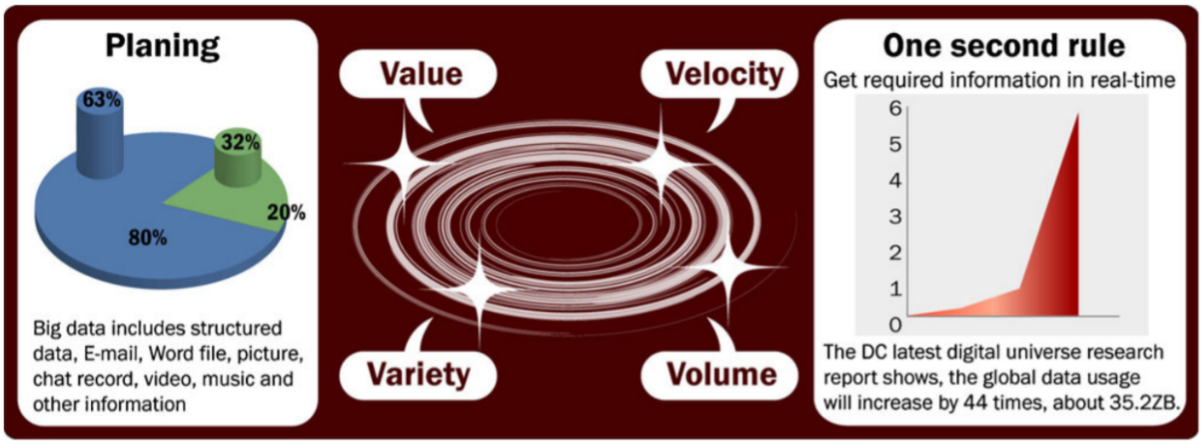


Figure 2.2: The four dimensions of big data according to IDC: Volume, Velocity, Variety, and Value. *Source: [CML14].*

quickly and timely conducted in order to properly process data, mine interesting insights and exploit their commercial value.

Variety Data sources are heterogeneous. Beside traditional structured data, e.g. relational databases, semi-structured as well as unstructured data emerge, such as text, audio, video, web content, etc.

The 3Vs model was not originally used to define big data, but many companies, including Gartner [Bey11], IBM [ZE+11], and Microsoft [Mei11], continued to employ the "3Vs" model to describe big data within the following ten years.

In 2011, the global consulting agency McKinsey & Company announced big data as the next frontier for innovation, competition, and productivity [MCB+11a]. According to their definition, the expression "big data" refers to datasets that could not be acquired, stored, and managed by classic database software. This means that dataset volumes considered as big data are continuously changing, possibly growing over time or due to technological advances.

In the same year, an IDC report defined big data as a new generation of technologies and architectures, designed to economically extract value from very large volumes of a wide variety of data, by enabling a high-velocity capture, discovery, and/or analysis [GR11]. The IDC definition adds the Value dimension to the traditional 3Vs model, which turns into a 4Vs model, as shown in Figure 2.2. Mining values from large-scale, heterogeneous, and quickly-generated datasets is probably the most critical problem in big data scenarios.

Finally, NIST defines big data as follows: *"Big data shall mean the data of which the data volume, acquisition speed, or data representation limits the capacity of using traditional relational methods to conduct effective analysis or the data which may be effectively processed with important horizontal zoom technologies"*. Such a definition indicates that novel efficient as

well as effective methods or technologies need to be developed and used to analyze and process big data.

2.1.2 The value of big data

McKinsley & Company pointed out that big data may improve the productivity and competitiveness of enterprises and public sectors, and create benefits for customers. The value can be obtained through research on the core industries that represent the global economy, namely the U.S. healthcare and retail, the EU public sector administration, the global personal location data, and the global manufacturing.

In [MCB+11a], McKinsey declared that big data could be employed to improve efficiency and quality. The U.S. medical industry may exceed USD 300 billion, thus reducing the expenditure for the U.S. healthcare by over 8%; retailers may improve their profit by more than 60%; the developed economies in Europe could save over EUR 100 billion.

In [GMP+09], Google found that during the 2009 flu pandemic, entries frequently sought at its search engines were different than usual, and the usage frequencies of the entries were correlated to the influenza spreading in both time and location. Analyzing big data, Google found 45 search entry groups that were closely related to the outbreak of influenza, and included them in mathematical models to forecast the spreading of influenza in terms of both time and location.

According to a well-known survey [CML14], data has nowadays become an important production factor that could be comparable to material assets and human capital. As multimedia, social media, and IoT are developing, enterprises will collect more information, leading to an exponential growth of data volume. Big data will have a huge and increasing potential in creating values for businesses and consumers.

2.1.3 Open challenges

The enormous increase of data induces challenges on data collection, storage, management, processing and analysis. Traditional relational database management systems (RDBMSs) are no longer adequate to handle the huge volume and heterogeneity of big data. Indeed, RDBMSs are increasingly using more expensive hardware, and they only deal with structured data.

Some solutions have been proposed in literature to meet different requirements. Cloud computing provides an efficient and elastic infrastructure, facilitating resource upgrading or downgrading. Distributed file systems [HKM+88] and NoSQL [Cat11] databases offer storage and management for very large datasets. Big data applications can be developed based on these technologies or platforms, but several challenges should be taken into account during development and deployment. Challenges and opportunities, broadly discussed in [ABB+11, LJ12, CDN11], are summarized in a survey by Chen et al. [CML14], who identified the following aspects:

Data representation Data representation consists in finding the most appropriate format for data analysis and user interpretation. It should reflect the different levels of heterogeneity datasets have in structure, type, organization, granularity, semantics, and accessibility. Efficient representation allows exploiting information richness and mining valuable insights.

Redundancy reduction and data compression Redundancy reduction and data compression are useful techniques to reduce the cost of a system, in particular from the computational point of view. Such approaches are especially relevant for datasets with high level of redundancy, and only if values are not affected by the process. A misapplication of reduction and compression techniques may entail a loss of information.

Data life cycle management Data life cycle management is a policy-based approach to manage the data flow of an information system throughout its life cycle, i.e. from creation and initial storage to obsolescence and deletion. Such mechanisms are required, since storage systems could not support the massive data that are generated by sensors, social media, and other data sources.

Analytical mechanism A well-devised analytical procedure is necessary to timely process large-scale heterogeneous data. Differently from RDBMSs, which typically are not scalable and are employed in the processing of structured data, non-relational databases have shown advantages in the processing of unstructured data. Compromise solutions are often required that integrate the benefits of RDBMSs and non-relational databases. In-memory databases as well as sample data based on approximate analysis are hot research topics.

Data confidentiality Data confidentiality is among the most critical topics to date, in particular after General Data Protection Regulation [gdp18] has been approved in EU and the scandal that involved Cambridge Analytica [Cad17] came out. Big data service providers frequently rely on third party data analysis tools to deal with their limited storage and computational capacity. This data processing increases the potential safety risks, since sensitive information may be present within data. Proper preventive measures should be taken to protect sensitive data and ensure its safety.

Energy management Energy management has recently drawn attention with the increase of data volume and analytical demands. The processing, storage, and transmission of massive datasets bring about more power consumption; as a consequence, efficient energy management and control mechanisms are needed.

Expendability and scalability Algorithms should be able to process increasingly expanding and more complex datasets. Scalability and the capability to seamlessly learn from new examples are key characteristics that big data algorithms should have.

Cooperation Cooperation is required to fully exploit the potential of big data, because of its interdisciplinariety. A comprehensive big data network architecture is advisable to help experts cooperating in order to reach common analytical objectives.

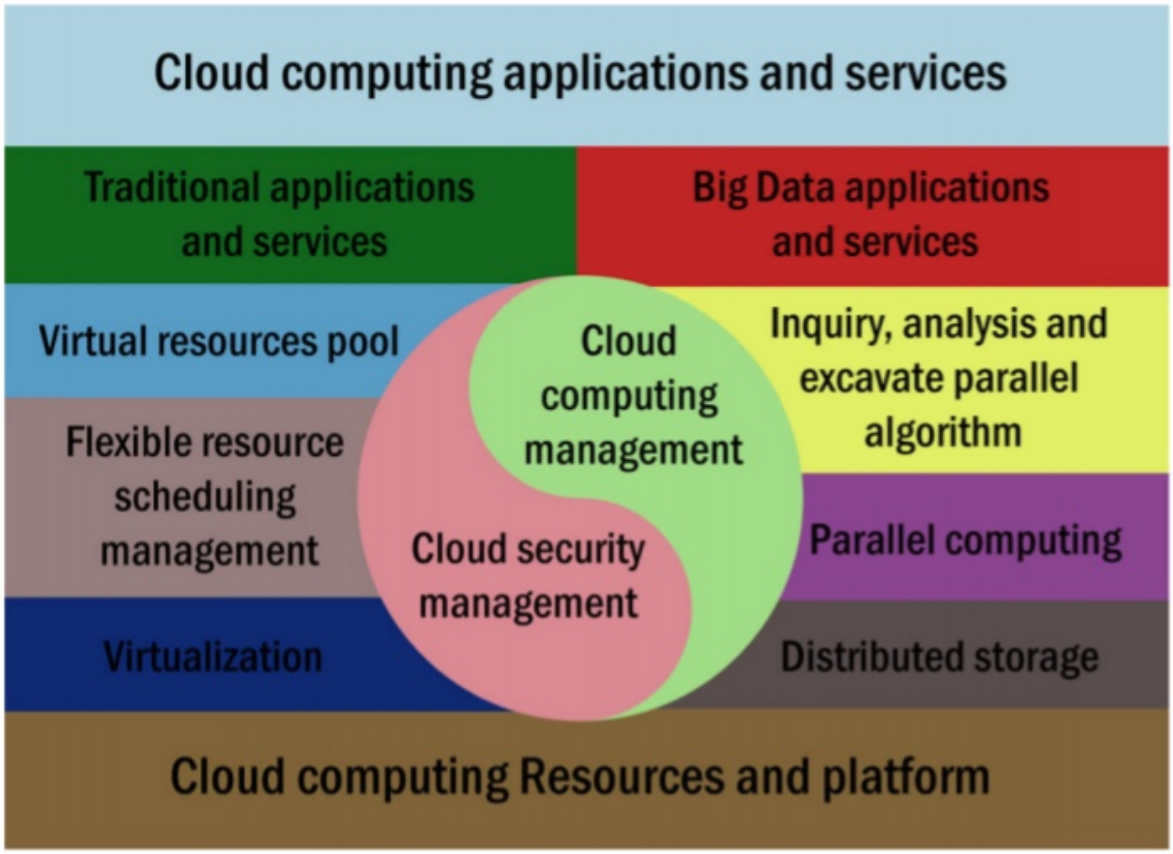


Figure 2.3: Main components of cloud computing. *Source: [CML14].*

2.2 Related technologies

The phenomenon of Big Data is facilitated by other outstanding technologies, including cloud computing, IoT, data mining and machine learning. Such technologies support the evolution of large-scale data and contribute to the extraction of relevant information from massive heterogeneous data sources. These recent advances are briefly introduced, giving special attention to their relationships with big data, so as for readers to gain a deeper understanding of current technological trends.

2.2.1 Cloud computing

Cloud computing is an IT paradigm that enables ubiquitous access, usually over the Internet, to a shared pool of configurable resources and higher-level services that can rapidly be provisioned with minimal management effort.

The macro components of cloud computing are shown in Figure 2.3. Cloud computing allows

using huge distributed computing and storage resources with minimal concentrated management. Big data can be subject to computationally intensive operations, and make use of the storage capacity of cloud systems. The distributed storage technology based on cloud computing can effectively manage big data, whereas the parallel computing capacity can improve the efficiency of collecting and analyzing big data thanks to cloud computing.

Although cloud computing and big data are closely related technologies, they have different purposes. The former is a new way to conceive an IT architecture, whereas the latter impacts on the decision-making process. As a consequence, the target customers of cloud computing are Chief Information Officers (CIO) focusing on IT solutions, whereas the target customers of big data are Chief Executive Officers (CEO) focusing on business. In spite of having different objectives, the evolution of these two technologies is positively correlated, because the growth of cloud computing from virtualized technologies provides the necessary infrastructure for big data processing.

2.2.2 IoT

The name Internet of Things (IoT) refers to the spread of embedded systems, devices, machines that are connected with each other through some kind of communication media, e.g. NFC, Bluetooth, the Internet, etc. Mobile devices, electrical appliances, public facilities are examples of such systems that are able to collect and transfer data, as shown in Figure 2.4. Devices are typically provided with sensors and actuators, respectively to perceive and affect the state of the environment wherein they are located. Different kinds of data can be gathered through sensors, such as location data (e.g. latitude, longitude, distance), environmental data (e.g. temperature, humidity, pressure), traffic data, and so on.

Different IoT devices exist that are used for disparate purposes; consequently, the collected data are typically heterogeneous. The IoT data are also unstructured and redundant, because values are periodically sensed and transmitted in raw form. Moreover, their physical nature makes sensors producing data that are inherently affected by noise.

According to HP, the quantity of sensors will reach one trillion by 2030 and then the IoT data will become the most important source of big data [CML14]. A report from Intel pointed out that the IoT data have three features that conform to the big data paradigm:

- abundant terminals generating masses of data;
- data generated by the IoT are usually semi-structured or unstructured;
- the IoT data are useful only when they are analyzed.

Despite their relatedness, the diffusion of the IoT is mainly due to the effective integration of big data and cloud computing. The IoT is a fundamental factor in the growth of data that will be a great opportunity for the application and development of big data, which in turn will boost research advances in the IoT.

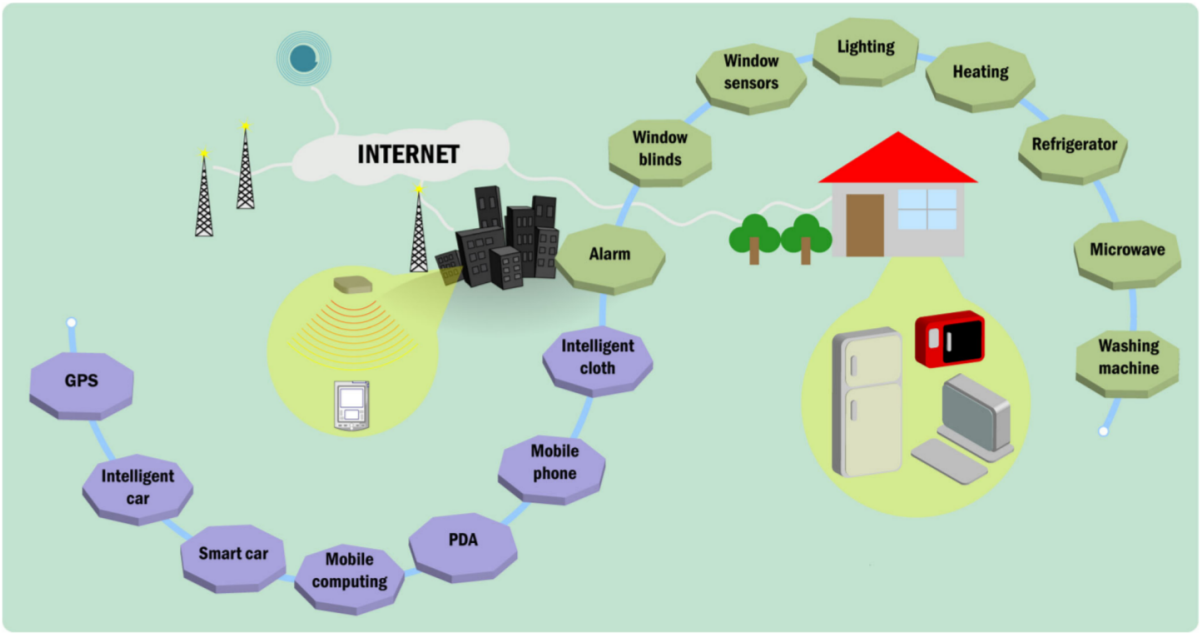


Figure 2.4: Data acquisition equipment in IoT. *Source: [CML14].*

2.2.3 Data mining and machine learning

Data mining is an interdisciplinary subfield of computer science, providing techniques and methodologies that allow the discovery of interesting, unexpected or valuable structures in large datasets [WFHP16]. Data mining takes advantage of statistical techniques, information retrieval, and machine learning for data processing and analysis.

Machine learning (ML) is a branch of artificial intelligence (AI) that focuses on the development of computer programs that can access data and automatically learn from them, without being explicitly programmed [Alp09]. Learning from examples allows machines performing complex tasks without human intervention, sometimes with better performance. The general ML workflow can be seen in Figure 2.5.

Machine learning tasks and algorithms are generally divided in several categories depending on the type of learning process:

Supervised learning Supervised learning entails the use of past labeled examples to learn a model that is able to predict future events. In particular, a function is inferred from a known training set, then applied in order to make predictions about the events of interest (i.e. the output values).

Unsupervised learning Unsupervised learning is used when the information used to train the model is unlabeled. Unsupervised approaches try to find a function that extracts a hidden structure from unlabeled data, without figuring out the right output.

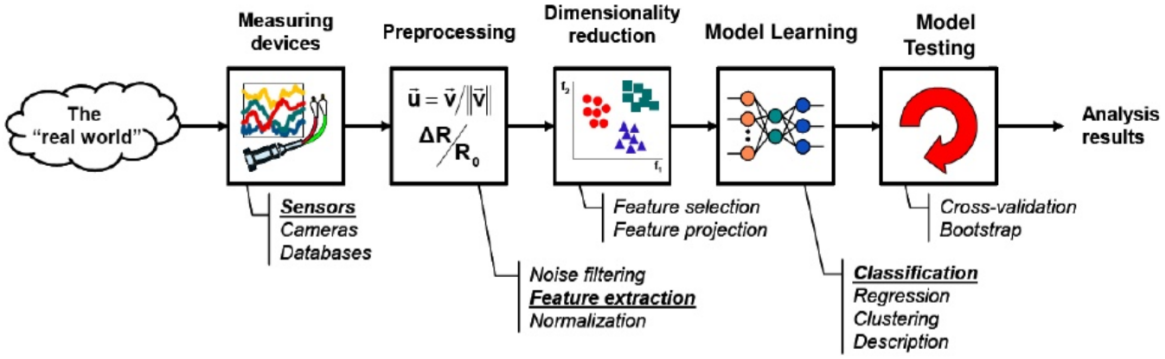


Figure 2.5: The ML process. *Source: [Rok12].*

Semi-supervised learning Semi-supervised algorithms make use of both labeled and unlabeled data to train a model. They are particularly useful when few labeled data are available and labeling new examples is a costly task.

Reinforcement learning Reinforcement learning is a learning method based on trial and error search and delayed reward. Basically, an agent interacts with the environment, performing actions and receiving real-valued rewards. Such rewards are reinforcement signals that are used by agents to identify their ideal behavior, that is, the policy that maximizes future rewards.

Conventional machine learning techniques have a limited ability to process raw data. For decades, constructing a pattern recognition or machine learning system required careful engineering. Considerable domain expertise was needed to design a feature extractor that transformed raw data (e.g. the pixel values of an image) into a suitable internal representation or feature vector from which the learning subsystem, often a classifier, could detect or classify patterns in the input [LBH15].

With the advent of representation learning methods as deep learning, the representations needed to perform complex tasks are automatically learned from data. The huge amount of information available in big data, and the computational support provided by cloud computing, accelerate the progress of machine learning algorithms and architectures. More accurate functions, better describing phenomena of interest, can be extracted by mining massive datasets with the help of cloud resources.

3
Sentiment analysis

This chapter gives an overview of sentiment analysis, which is the first text mining topic this thesis contributes to in Part II. Readers are provided with a background of the task, including its rationale, main analytical techniques, and utility in real world scenarios.

Specifically, the task is introduced, emphasizing the aspects that make it a compelling research thread and pointing out its practical implications. Afterwards, some notable sentiment analysis techniques that are relevant to this thesis are described, namely feature selection, sentiment classification, and transfer learning in cross-domain tasks.

3.1 Introduction

Sentiment Analysis (SA), also known as Opinion Mining (OM), is the computational study of people's opinions, appraisals, attitudes, and emotions toward an entity (surveys are in [LZ12, Liu12, PL+08, VC12, MHK14]). The entity can represent individuals, events, or topics, which are most likely to be covered by reviews. Although some researchers stated that OM and SA have slightly different notions [TP12], the former extracting and analyzing people's opinion about an entity while the latter identifying the sentiment expressed in a text then analyzing it, the two expressions can mostly be considered as interchangeable.

Sentiment analysis is a challenging, but very useful task in practice. Understanding people's opinions about products, services, brands and so on is of valuable importance for operational as well as strategic business decisions. Companies always want to know customers' opinions about their products and services to identify what people are looking for, to establish which features to add next, or to find the reason behind a complaint.

As illustrated in Figure 3.1, sentiment analysis can be considered a classification process. There are three main classification levels in sentiment analysis, namely document-level, sentence-level, and aspect-level. Document-level sentiment analysis aims at classifying an opinion document as expressing a positive or negative opinion or sentiment. The assumption is that a document is a basic information unit that discusses about only one topic. Sometimes this hypothesis is too restrictive, since opinions on different entities can be expressed in a document. However, a document-level analysis is very useful to identify the overall sentiment orientation, or polarity, of a plain text.

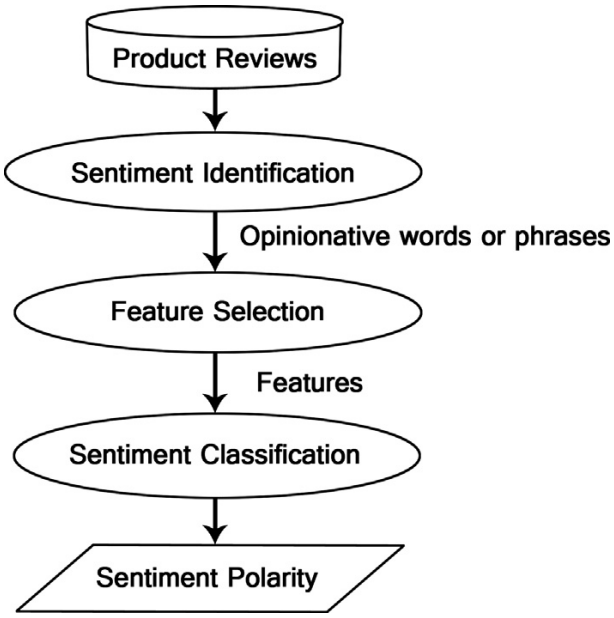


Figure 3.1: Sentiment analysis process on product reviews. *Source: [MHK14].*

In order to classify the polarity of each sentence, sentence-level sentiment analysis si required. The first-step is to identify whether a sentence is subjective or objective. In case it is subjective, sentence-level sentiment analysis will determine its sentiment orientation. However, Wilson et al. [WWH05] pointed out that subjective expressions do not necessarily have a polarity (i.e. they can also be neutral).

Liu [Liu12] argued that there is no fundamental difference between document and sentence level classifications, because sentences are just short documents. Classifying text at the document level or at the sentence level does not require to extract opinions on all aspects of the entity involved. In case detailed opinions about specific aspects of entities are needed, aspect-level sentiment analysis is involved. The first step is to identify the entities and their aspects. The opinion holders can give different opinions for different aspects of the same sentence, e.g. *"The actors of the movie are very good, but I don't like the plot."*

The data sets used for sentiment analysis primarily are from product reviews, whose sources typically are review sites. However, sentiment analysis is not only applied on product reviews but also on stock market [YWCC13, HLN13], news articles [XPC12], or political debates [MV12]. For example, people's opinions on certain election candidates or political parties can be identified within a political debate. The election results can also be predicted from political posts. Social networks and micro-blogging sites are considered a very good source of information because people freely share and discuss opinions about a certain topic.

In order to build a model that is able to identify the sentiment orientation of a plain text, a set of documents is required. Text representation methods are applied to map raw data into a more suitable representation, for instance the bag-of-words model, where documents are transformed

into a document-term matrix [ZJZ10], whose entries are functions of the word occurrences in documents. Before such a transformation, documents are generally denoised using standard pre-processing steps, such as word tokenization, case folding, number removal, stopwords removal. Stemming [Por80] as well as lemmatization [PLM+04] are sometimes applied.

3.2 Techniques

Many applications and enhancements on sentiment analysis have been proposed in the last few years. Techniques can be divided into logically different subareas, namely Feature Selection (FS), Sentiment Classification (SC), Emotion Detection (ED), Building Resources (BR), and Transfer Learning (TL). Feature selection approaches allow to identify and choose only the features that are supposed to be promising for classification. Sentiment classification copes with the problem of automatically organizing a corpus of documents into a predefined set of categories, or classes, according to their polarity, e.g. positive, negative, neutral. Emotion detection aims at extracting and analyzing emotions in sentences, which could be either explicit or implicit. Building resources consists in creating lexica, corpora in which opinion expressions are annotated according to their polarity, sometimes relying on dictionaries. Transfer learning and cross-domain classification are concerned with analyzing data from one domain and then using this knowledge in a heterogeneous target domain.

Feature selection, sentiment classification, and transfer learning will be further discussed in this section, pointing out their rationale, operations, and practical utilities, and reviewing the main related research advances. On the other hand, the contributions of this thesis based on such techniques will be introduced in Part II.

3.2.1 Feature selection

Bag-of-words models are inherently affected by the curse of dimensionality [Bel13], because every term in the analyzed corpus becomes a feature of the model. As a consequence, the representation also suffers from the data sparsity problem, which affects both model efficiency and effectiveness. To address these issues, only the most relevant features are selected. Examples of features that usually support the identification of sentiment are [AZ12]:

Terms presence and frequency These features are individual words or word n-grams, and their frequency counts. Binary weighting as well as term frequency weights can be used to indicate the relative importance of features [MS11].

Parts of speech (POS) Part-of-speech tagging maps each word to a part of speech (e.g. nouns, attributes, verbs, etc.). Attributes foster opinion detection and discovery, because they frequently are polarity-bearing terms (e.g. *good*, *bad*). However, nouns (e.g. *disaster*, *miracle*), verbs (e.g. *accomplish*, *fail*), conjunctions (e.g. *but*) and negations (e.g. *not*) can also help to identify the polarity.

Opinion words and phrases Opinion words are terms (i.e. parts of speech) commonly used to express opinions, such as *good* or *bad*, *love* or *hate*, *great* or *awful*. Some phrases may express opinions without using opinion words, e.g. *cost me an arm and a leg*.

Negations Negations are particular parts of speech that may change the sentiment orientation, e.g. *not good* is more similar to *bad* than *good*.

Feature selection methods can be divided into lexicon-based methods and statistical methods. The former require human annotation: they usually start with a small set of "seed" words, which are then expanded through synonym detection or online resources to obtain larger lexica. On the other hand, the latter are fully automatic approaches, widely used in sentiment analysis. Examples of statistical feature selection approaches are:

Point-wise Mutual Information (PMI) Point-wise mutual information provides a formal way to represent the mutual information between features and classes. The point-wise mutual information $M_i(w)$ between word w and class i depends on their co-occurrence. The expected co-occurrence of class i and word w is given by $F(w) \cdot P_i$, whereas the true co-occurrence is given by $F(w) \cdot p_i(w)$. The mutual information is given by the following equation:

$$M_i(w) = \log\left(\frac{F(w) \cdot p_i(w)}{F(w) \cdot P_i}\right) = \log\left(\frac{p_i(w)}{P_i}\right) \tag{3.1}$$

$M_i(w)$ is greater than 0 when word w is positively correlated to class i, whereas $M_i(w)$ is less than 0 in case of a negative correlation.

Yu et al. [YWCC13] extended PMI by developing a contextual entropy model to expand a set of seed words. Such a model discovers other words similar to the seed words, by comparing the contextual distribution between each couple of words using an entropy measure. Considering both co-occurrence strength and contextual distribution is beneficial for sentiment classification, because it allows acquiring more useful emotion words and fewer noisy words.

Chi-square $\left(\chi^2\right)$ Chi-square statistic between word w and class i is defined as in [AZ12]:

$$\chi_i^2 = \frac{n \cdot F(w)^2 \cdot (p_i(w) - P_i)^2}{F(w) \cdot (1 - F(w)) \cdot P_i \cdot (1 - P_i)} \tag{3.2}$$

where n is the total number of documents in the collection, $p_i(w)$ is the conditional probability of class i for documents that contain w, P_i is the global fraction of documents containing class i, and $F(w)$ is the global fraction of documents that contain word w.

Applications of chi-square include contextual advertising, to identify ads that are positively correlated with a blogger's personal interests [FC11], and stock market data, where feedback features obtained by capturing market feedback are used as a part of the feature selection process [HLN13].

Latent Semantic Indexing (LSI) Latent semantic indexing is a feature transformation method that maps the text space to a new axis system, which is a linear combination of the original features [DDF+90]. This goal is achieved through Principal Component Analysis techniques as Singular Value Decomposition (SVD) [Jol11], an unsupervised method that determines the axis-system that retains the greatest level of information about the variations in the underlying attribute values.

Other statistical methods exist apart from those just mentioned, including information gain and Gini index [AZ12], Hidden Markov Models (HMMs) and Latent Dirichlet Allocation (LDA). The last two techniques have been used in [DS12] to separate the entities in a review document from the subjective expressions that describe those entities in terms of polarities, and combined in [GSBT05] to simultaneously model topics and syntantic structures within a corpus.

One of the most challenging tasks where feature selection plays an important role is irony detection. For this purpose, Reyes and Rosso [RR12] defined a feature model in order to represent part of the subjective knowledge within reviews, and tried to describe salient characteristics of irony. Six categories of features were taken into account to model irony, such as n-grams, POS-grams, funny profiling, positive/negative profiling, affective profiling, and pleasantness profiling. Using naïve bayes, support vector machine, and decision tree as sentiment classifiers, they achieved good results in terms of accuracy, precision, recall, and F-measure.

3.2.2 Sentiment classification

In general, text classification copes with the problem of automatically organizing a corpus of documents into a predefined set of categories or classes, which usually are the document topics, like for instance sport, politics, cinema and so on. Sentiment classification is a particular text classification task that distinguishes documents based on their polarity. The polarity, or sentiment orientation, of a plain text is typically denoted by either a label (e.g. positive, negative, or possibly neutral) or a numeric value (e.g. a score from 1 to 5).

This supervised approach learns a classification model from a training set of documents, labeled according to their sentiment orientations, in order to classify new unlabeled documents, whose polarities have to be discovered. The more new documents reflect the peculiarity of the training set, the more the classification of the test set will be accurate. Generally, the parameters of the learning algorithm adopted can be tuned on a validation set of documents, labeled as the training set, in order to maximize the performance on the test set.

The semantic comprehension of natural language text is arduous, because of its intrinsic ambiguity and context dependence. Both word polarity, namely its positive or negative orientation, and relationships among words have to be taken into account to properly understand the meaning of a sentence. The task becomes even more challenging when document-level understanding is required, namely when the overall document polarity has to be discovered.

The typical approach to sentiment classification assumes that both the training set and the test set deal with the same topic. For example, a model is learned on a set of board game reviews

and applied to a distinct set of reviews, but always about board games. This modus operandi, known as in-domain sentiment classification, guarantees optimal performance provided that documents from the same domain are semantically similar. Nevertheless, the implicit assumption in-domain classification relies on is to have labeled documents from the same domain of the text set whose sentiment orientation has to be found. Unluckily, given that most documents are usually unlabeled, the in-domain approach is often inapplicable in practice. Tweets, blogs, fora, chats, emails, public repositories, social networks are sources of unsupervised data that could bear opinions, but no information is available on whether such opinions are positive, negative or neutral.

A really simple solution to the absence of labeled data is to manually pre-classify documents, so as to have enough labeled data to train an in-domain sentiment classifier. This method is expensive and not scalable with very large text sets; therefore, it cannot be used in big data scenarios.

This is the reason why cross-domain solutions are needed, where a model can be reused across different domains. A deeper discussion on cross-domain sentiment classification and transfer learning techniques will be provided in Section 3.2.3.

Sentiment classification techniques

Sentiment classification techniques, whose overview can be found in Figure 3.2, can be distinguished according to the approach used. Macro categories are machine learning approaches, lexicon-based approaches and hybrid approaches [MF11].

The Machine Learning (ML) approach relies on ML algorithms that treat sentiment analysis as a text classification task, where the polarity of plain text is identified by means of syntactic and/or linguistic features. The lexicon-based approach, which includes dictionary-based and corpus-based methods, relies on a sentiment lexicon, that is a collection of known and precompiled sentiment terms. The hybrid approach simply is the combination of the previous two. Lexicon having a well-known polarity is frequently used together with traditional machine learning algorithms.

In a sentiment classification task, we have a set of training instances $D = \{X_1, X_2, \ldots, X_n\}$, where each example is labeled with a sentiment orientation, which can be either a score or a nominal tag identifying the polarity. Classification algorithms first learn a mapping function from instances to classes, then are used to predict a label for each new unlabeled example. The approach just mentioned is an example of supervised learning, which is applicable anytime enough labeled training instances are available to train a robust model.

Many supervised classifiers are used in sentiment analysis. A first category are probabilistic classifiers, which use mixture models for classification, assuming that each class is a component of the mixture. Each mixture component is a generative model that provides the probability of sampling a particular term for that component.

Probabilistic classifiers, sometimes referred as generative classifiers, include Naïve Bayes, Bayesian Network, and Maximum Entropy Classifier. Naïve Bayes computes the posterior

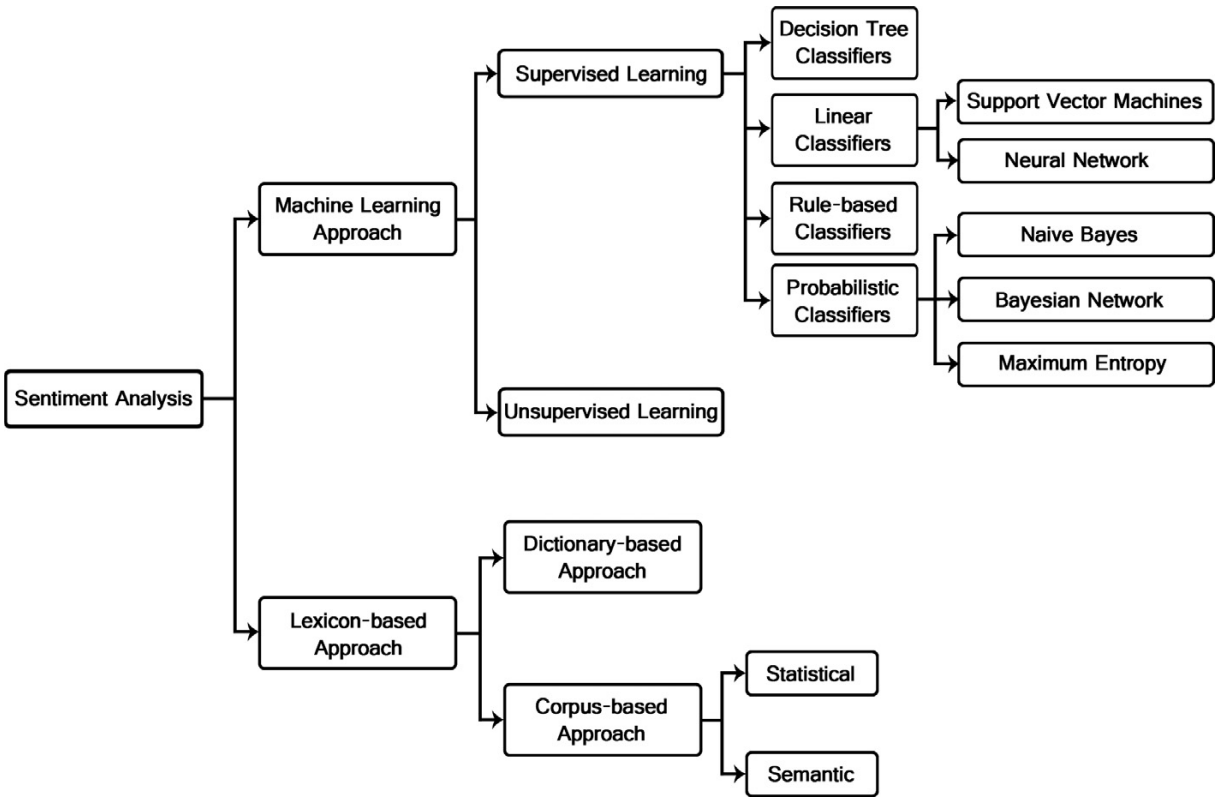


Figure 3.2: Sentiment classification techniques. *Source: [MHK14].*

probability of a class, based on the distribution of the words in the document. Each feature of the model is assumed to be conditionally independent of others. On the other hand, bayesian network assumes that the features are fully dependent, specifying a complete joint probability distribution over all variables. Maximum entropy classifier encodes labeled feature sets to vectors. The encoded vector is used to compute weights for each feature that can then be combined to determine the most likely label for a feature set. This classifier is parameterized by a set of weights, which is used to combine the joint features that are generated from a feature-set by an encoding.

A second category of classifiers are linear classifiers, such as Support Vector Machine and Neural Network. Support vector machine aims at determining the linear separators in the search space that can best divide classes. SVM is particularly suitable for text data because of the sparse nature of text, in which few features are irrelevant, but they tend to be correlated with one another and generally organized into linearly separable categories [Joa96]. SVM can also divide nonlinearly separable instances. Relying on kernel functions, data are mapped from the original feature space into a new space where the classes can be linearly separated with a hyperplane [Aiz64]. On the other hand, a neural network consists of many neurons where the neuron is the basic unit. Neurons can compute a function of their inputs, which typically are word frequencies as well as a set of weights. Neural networks can be composed of many layers to deal with nonlinear surfaces, where the outputs of the neurons in the earlier layers feed into the neurons in the later layers. Neural networks are trained by back-propagating errors over different layers [RHW86]. Researchers have already been employed neural networks for text data [RS99, NGL97], and also compared support vector machines and artificial neural networks [MVN13].

A third category of classifiers are decision trees, which recursively decompose the training data space according to conditions on the attribute values [Qui86]. The condition may be the presence or absence of some words, or a value interval for some attributes. The division of the data space is done recursively until the leaf nodes are eventually reached. The main difference among decision tree approaches is the criterion used to split data, for example single attribute [LR94], similarity-based multi-attribute and discriminant-based multi-attribute [CRS03]. The decision tree implementations in text classification tend to be small variations on standard packages such as ID3 and C4.5.

The last category of classifiers are rule-based classifiers, which label data according to a predefined set of rules. The left hand side typically represents a condition on the feature set, whereas the right hand side represents the class. The two most frequently used criteria to generate rules are support and confidence [ML98].

Beyond the just mentioned approaches, meta classifiers are also used to learn more robust models. Among the most known meta classifiers methods there are bagging [Bre96], which generates multiple versions of a predictor and uses them to get an aggregate predictor, and boosting [FS97], which iteratively learns weak classifiers with respect to a distribution until a strong classifier is eventually obtained.

In any of the algorithms discussed, a large number of labeled training documents is needed

to learn a robust sentiment classifier. When labeled data are unavailable, either unsupervised techniques or transfer learning approaches are employed to identify the sentiment orientation of a set of documents. Transfer learning techniques will be discussed in Section 3.2.3.

Among the unsupervised methods, Turney [Tur02] introduced an algorithm to evaluate mutual information between the given sentence and two words taken as reference: "excellent" and "poor". Furthermore, Taboada et al. [TBT+11] built a dictionary of words annotated with both their semantic polarities and their weights, also including intensification and negation. These are examples of lexicon-based techniques.

The lexicon-based approach is the other primarily used approach for sentiment classification. It basically consists in finding a set of opinion words, which can help identifying the sentiment orientation of documents.

For this purpose, a first commonly used strategy is the dictionary-based [KH04, HL04]. A small set of opinion words with known orientations is manually constructed, then synonyms and antonyms are added by searching in well-known corpora as WordNet [Mil95] or in thesaurus [MDD09]. The newly found words are iteratively added, and the process stops when no new words are found. The main drawback of the dictionary-based strategy is its inability to find opinion words with domain and context specific orientations. To solve this problem, the corpus-based strategy is adopted.

Corpus-based methods generally depend on syntactic patterns or patterns that occur together, along with a seed list of opinion words to find other opinion words in a large corpus. Although this approch can help finding domain and context specific opinion words, it is not as effective as the dictionary-based if used alone, because it is hard to have a huge corpus covering all English words.

Natural language processing (NLP) techniques are often used with the lexicon-based approach to find syntactical structures and mining semantic relations [BG04]. Such techniques have been employed for the identification of tense and time expressions [MP12], to analyze sentences by means of a dependency parsing [DCG13], and as a pre-processing stage before the application of a lexicon-based sentiment analysis algorithm [MRCZ12].

Most methods usually employed hybrid approaches to take advantage of both the machine learning and the lexicon-based approaches. For example, Dave et al. [DLP03] draw on information retrieval methods for feature extraction, and to build a scoring function based on words found in positive and negative reviews. In [TWC08, QZHZ09], a dictionary containing commonly used words in expressing sentiment is employed to label a portion of informative examples from a given domain. These instances reduce the labeling effort and allow using the labeled documents as a training set to learn a supervised classifier. Melville et al. [MGL09] showed that lexical information about associations between words and classes can be exploited and refined for specific domains by means of training examples to enhance accuracy.

Finally, term weighting could also foster sentiment classification, as shown by some researchers that propose different term weighting schemes. Notable examples are a variant of the well-known tf-idf [PT10], a supervised scheme based on both the importance of a term in a document and the importance of a term in expressing sentiment [DLY14], the usage of

regularized entropy in combination with singular term cutting and bias term in order to reduce the over-weighting issue [WG14].

3.2.3 Cross-domain and transfer learning

Cross-domain learning is a very attractive research thread that paves the way for the use of large-scale unsupervised data. Basically, a model is learned on a labeled source domain, then used to classify the polarity of a distinct unlabeled target domain. For example, let us suppose to have learned a model on a set of reviews about movies, and now to be interested in understanding people's thoughts about kitchen appliances, whose reviews unfortunately are not labeled. A manual categorization of the unlabeled reviews can be a time-consuming solution, infeasible with large-scale text sets. Instead of manually labeling the kitchen appliances' reviews and building a model from scratch, we want to exploit a model that is already available.

The rationale behind the cross-domain approach is the fact that it allows for model reuse: a model is built once, and used multiple times. The reusability of a model is relevant for both research and business, since collecting, labeling, processing and analyzing data can really be time-consuming tasks, which hinder scalability in big data scenarios. However, the test documents may not reflect the regularities of the training set due to the heterogeneity of language. For example, if a movie review is likely to include terms like *amusing* or *boring*, a kitchen appliance review is more plausible to contain *clean* or *broken*. Hence, a transfer learning phase is typically demanded to bridge the inter-domain semantic gap.

Transfer learning generally entails learning knowledge from a source domain and using it in a target domain. Specifically, cross-domain methods are used to handle data of a target domain where labeled instances are only available in a source domain, similar but not equal to the target one. While these methods are used in image matching [SMGE11], genomic prediction [DMMP16] and many other contexts, classification of text documents by either topic or sentiment is perhaps their most common application.

Two major approaches can be distinguished in cross-domain classification [PY+10]: "instance transfer" directly adjusts source instances to the target domain, while "feature representation transfer" maps features of both domains to a different common space. The most popular transfer learning methods are clustering algorithms and approaches that make use of feature expansion and external, sometimes hierarchical, knowledge bases. These techniques, employed in conjunction with standard text classification algorithms, lead to good results in sentiment classification [TWC08, QZHZ09, MGL09]. Nevertheless, a tuning of parameters is necessary to ensure adequate effectiveness, because the parameter values that yield optimal accuracy with a text set do not usually produce analogous best results with different corpora. Parameter tuning is a bottleneck anytime a new text set has to be classified. Defining algorithms that are not affected (or slightly affected) by such a problem represents an open research challenge.

With reference to text categorization by topic, transfer learning has been fulfilled in different ways, for example by clustering together documents and words [DXYY07], by extending probabilistic latent semantic analysis to unlabeled instances [XDYY08], by extracting latent

words and topics, both common and domain specific [LJL12], by iteratively refining target categories representation without a burdensome parameter tuning [DMPS14a, DMPS14b].

Many transfer learning methods have also been proposed for cross-domain sentiment classification. Aue and Gamon [AG05] tried some approaches to customize a classifier to a new target domain: training on a mixture of labeled data from other domains where such data are available, possibly considering just the features observed in target domain; using multiple classifiers trained on labeled data from diverse domains; including a small amount of labeled data from target. Bollegala et al. [BWC13] suggested the adoption of a thesaurus containing labeled data from source domain and unlabeled data from both source and target domains. Blitzer et al. [BDP07] discovered a measure of domain similarity contributing to a better domain adaptation. Pan et al. [PNS$^+$10] advanced a spectral feature alignment algorithm which aims to align words belonging to different domains into same clusters, by means of domain-independent terms. Such clusters form a latent space which can be used to improve sentiment classification accuracy of a target domain. He et al. [HLA11] extended the joint sentiment-topic model by adding prior words sentiment, thanks to the modification of the topic-word Dirichlet priors. Feature and document expansions are performed through adding polarity-bearing topics to align domains. Zhang et al. [ZHL$^+$15] proposed an algorithm that transfers the polarity of features from the source domain to the target domain with the independent features as a bridge. Their approach focused not only on the feature divergence issue, namely different features are used to express similar sentiment in different domains, but also on the polarity divergence problem, where the same feature is used to express different sentiment in heterogeneous domains. Franco et al. [FSCTR15] used the BabelNet multilingual semantic network to generate features derived from word sense disambiguation and vocabulary expansion that can help both in-domain and cross-domain tasks. Bollegala et al. [BMG16] modeled cross-domain sentiment classification as embedding learning, using objective functions that capture domain-independent features, label constraints in the source documents and some geometric properties derived from both domains without supervision.

4
Stock market analysis

It is no mistery that stock market analysis is a primary interest for finance, and has historically attracted interest from shareholders as well as academia. Understanding the market trend allows investors choosing the best trade-off strategy between risk minimization and profit maximization.

This chapter gives an overview of the problem, focusing on the techniques used for analysis and prediction rather than on the financial point of view. In particular, several renowned approaches for stock market forecasting are discussed, emphasizing the predictive potential of social media. Driven by such existing researches, a novel Twitter-based method for stock market prediction and trading will be proposed in Part III.

4.1 Introduction

The Efficient Market Hypotesis (EMH), proposed by Fama [Fam65], states that prices of financial assets are managed by rational investors who rely on new information, i.e. news. Since news is not predictable, neither is the stock market, which generally follows a random walk trend, according to past studies [KAYT90, FAM91]. The EMH was confuted by Malkiel [Mal03], who provided evidences that market prices reflect all the available information. Moreover, several studies showed that the trend of the stock market does not follow a random walk model and can be predicted in some way [LM88, BM92], for example applying mining techniques to market news [GE01, SC06], to past prices [LWD+11], or even to financial reports [LLKC11].

Many approaches to stock market analysis and prediction have been proposed over time, from time series prediction to textual news analysis and social network analysis. In this chapter, we start from traditional approaches to stock market prediction, then we review methods based on text and news analysis, and finally we summarize the most recent works using social network information to forecast the market prices.

4.2 Traditional approaches

Initially, the scientific researches were based on the Efficient Market Hypothesis (EMH) [Fam65], according to which the prices of traded assets reflect all relevant information available at any

time. In such a financial market model, neither technical prediction analysis of future prices based on the study of past prices, nor fundamental analysis studying the evolution of the business value, allow an investor to achieve higher profits than those that another investor would get with a randomly selected portfolio of stocks, with the same degree of risk.

The EMH is divided into three basic levels, namely "weak", "semi-strong" and "strong". The weak form implies that the market is efficient, reflecting all market information at a given time. According to this hypothesis, past rates of return on the market have no effect on future rates. The weak form invalidates the rules used by traders to buy/sell a stock. The semi-strong form incorporates the weak version with the addition that the information is immediately reflected in the price, becoming available to the public. Given this assumption, a trader cannot exploit new information to get advantages over and above the market. Finally, the strong form incorporates the previous hypotheses by assuming that the market reflects not only public information, but also private information. As a consequence, no investor would be able to profit the average investor even if she was given new information. Based on the EMH hypothesis, several researches [KAYT90, FAM91, Mal96] argued that the financial market, being driven by news, evolves in an unpredictable way according to the random walk theory.

In the last decades, a great amount of works refused the unpredictability hypothesis (a complete overview is given in [Mal03]), showing that stock price series follow a random behavior only in short periods of time, whereas they are predictable in general [LM88, BM92, QR07]. Research in behavioral finance [BT03, Mal03] has shown that investors give different psychological interpretations to new information. Tetlock [Tet07] studied the correlation between communication media and the market, showing that a media's high pessimism predicts a downward pressure on the market, increasing the market trading volume. Moreover, the EMH assumes that the stock market is driven by news, which are unpredictable. However, in an experiment conducted by Lebaron et al. [LAP99], an artificial stock market, driven by simulated commercial agents, was created to study the behaviour of decision-making processes of traders based on the timely introduction of information. It emerged from the analysis that, if the information is assimilated slowly, thus having a longer time horizon, the simulations look like the actual behavior of shareholders. In other words, the experiment highlighted a time-lag between the arrival of new information and the reaction of investors in the stock market, so that news can be analyzed during this time frame and used to make predictions.

Motivated by such an interesting discovery, many researchers focused their studies on stock market prediction. Two major approaches exist: the first uses features derived from technical analysis, based on the history of stock index prices; the second predicts trends by means of related news and textual information. While surveys of such approaches are provided by Atsalakis and Valavanis [AV09], and by Mittermayer [Mit04] respectively, next section will focus on the latter.

4.3 Methods based on text and news analysis

Thomas and Sycara [TS00] predicted the stock prices by integrating textual information from web bulletin boards into trading rules derived by genetic algorithms, based on the trading volumes of the stocks concerned as well as on the number of messages posted on the web bulletin. Lavrenko et al. [LSL$^+$00] proposed to predict the intra-day stock price movements of a company stock by analyzing the contents of real-time news stories related to that company. Schumaker and Chen [SC06, SC09] investigated on 9,211 financial news articles and more than 10 million stock quotes covering S&P 500 index during a period of five weeks. They used supported vector machines to estimate a stock price 20 minutes after the release of some news, getting the best results when the model is composed of both article terms and the index price at the release of the news. Mittermayer and Knolmayer [MK06b] introduced NewsCATS (News Categorization and Trading System) to predict stock price trends immediately after the publication of press releases. Their approach consists of three phases: (i) automatically pre-processing of incoming press releases, (ii) categorizing them into different news types and (iii) deriving trading rules for the corresponding stock. Fung et al. [FYL05] proposed a framework to predict the real-time movements of stock prices by analyzing news stories. The authors selected the positive training documents by means of χ^2, and combined them with the stock trends using a support vector machine. Several other papers pointed out the potential of financial news articles to create prediction models (a survey is in [MK06a]). For instance, Wuthrich et al. [WCL$^+$98] predicted stock price movements by counting the occurrences of predefined keywords in financial articles. Gidofalvi and Elkan [GE01] divided the stock price time series around the publication of a news article into windows of influence. During the learning phase, the documents are labeled according to a three-category model corresponding to increase, decrease or unchanged of price within the influence window. The first 1000 word features according to mutual information are automatically extracted, then a Naïve Bayes classifier is trained on them. Li et al. [LWD$^+$11] experimented different training methods and different lag times to understand the impact of news: news article only, ex ante prices only, and a naïve combination of news article and ex ante prices called multi-kernel learning (MKL). The tests were conducted with lag times from 5 to 30 minutes, and the best performance was achieved after 20 minutes from the news publication. Lin et al. [LLKC11] advanced a method called Hierarchical aglomerative and Recursive K-means clustering (HRK), in which qualitative and quantitative features are used to analyze financial reports. Moreover, the authors combined K-means and hierarchical clustering to detect clusters and improve quality of the prototypes of such financial reports. The prototypes made the proposed method noise tolerant and let it avoid overfitting problems. The experimental evaluation showed that HRK outperforms SVM and Naïve Bayes algorithms. Similarly, Zhai et al. [ZHH07] combined the information of news articles with technical indicators to predict the daily price movements of a stock. Lee [Lee09] developed a SVM-based model with hybrid feature selection and supported sequential forward search to predict the NASDAQ trends. Other researchers employed blog posts to predict stock market behaviour by determining the correlation between activities on Internet message boards, and stock volatility and trading volumes [AF04].

Gilbert and Karahalios [GK10] created an index of the US national mood, called ”anxiety index”, by exploiting over 20 million posts from the LiveJournal website. They noted that, when this index increases significantly, the S&P 500 ends the day marginally lower than expected. A comparative survey of artificial intelligence applications in finance is reported in [Bah10].

4.4 Methods based on social media

Social media are a pervasive global phenomenon, constantly increasing in the last decade. Facebook, Twitter, LinkedIn, Xing are among the most popular examples. They are huge data sources where users can insert information and opinions about various topics.

Since text and news have already been proved to help stock market analysis [Mit04, MK06a], researchers have started investigating whether social media can support prediction, being a valuable source of unstructured data. One of the most analyzed social networks is Twitter, which is very dense of informative content, because users must be very concise when expressing opinions, due to the fact that tweets cannot be longer than 140 characters.

4.4.1 The predictive value of Twitter

Twitter is a well-known American online news and social networking service on which users post and interact with messages known as ”tweets”. Twitter has over 330 million users and an average of 500 million tweets posted every day, offering the opportunity to access this stream of global posts. Considered the huge amount of data available, Twitter represents a huge knowledge base providing information about the most disparate topics.

Many research fields related to text mining have taken advantage of this source of information. For instance, sentiment analysis benefits from tweets due to the intrinsic nature of the information posted by users. Plenty of articles have been written on sentiment analysis and the study of tweet polarization [PP10, TBP11, AXV+11, KBQ14, KS12, BMMP13, MCMVULMR14]. It can be argued that such a knowledge base can provide an indication of the public mood. Indeed, the emotional state, as the prerogative of a single human being, propagates to social status as a feature of all individuals. This phenomenon was studied by Bollen et al. [BMP11], who found that events in the social, political, cultural and economic sphere do have a significant, immediate and highly specific effect on various dimensions of the public mood extracted from Twitter. They speculated that a large-scale mood analysis can provide a solid platform to model collective emotive trends in terms of their predictive value with regards to existing social as well as economic indicators. This predictive feature of Twitter mood has been used to forecast different phenomena, for example the sales of a movie [AH10], the public opinion on a particular brand [GSZ13], the political alignment of Twitter users [CRF+11, CGR+11], the results of elections [GA12].

4.4.2 Twitter-based stock market prediction

Many approaches in literature applied sentiment analysis techniques to tweets in order to create forecast models for the stock market. Bollen et al. [BMZ11] measured collective mood states (positive, negative, calm, alert, sure, vital, kind and happy) through sentiment analysis applied to more than 9 million tweets posted in 2008. Tweets were filtered by some generic sentiment expressions (e.g. "I'm feeling") that were not directly related to stock market. The authors analyzed tweets by two mood tracking tools: Opinion Finder (OF) [WHS+05], which classifies tweets as positive or negative, and Google-Profile of Mood States (GPOMS), which measures mood in six dimensions. They found that the *calm* mood profile yields the best prediction result for Dow Jones Industrial Average (DJIA) with accuracy of 86.7% in the prediction of daily directions in december. Furthermore, they showed how a tweet aggregation in a 3-day period ensures better prediction on the daily DJIA. Similarly, Chyan et al. [CHL12] used the *calm* score of tweets extracted from June and December 2009, achieving accuracy of 75% in a 20-day prediction of Dow Diamonds ETF (DIA). Accuracy increased up to 80% by adding a quantitative feature related to the previous DIA value.

An analysis similar to Bollen's one was conducted by Mittal and Goel [MG12]. The authors employed the same dataset used in [CHL12] for a multi-class classification, considering only *calm*, *happy*, *alert* and *kind* as mood dimensions. Furthermore, four learning algorithms (i.e. Linear Regression, Logistic Regression, SVMs and SOFNN) are used to learn and exploit the actual predictions; among the algorithms, SOFNN-based model performed best, achieving nearly 76% of accuracy. Oliveira et al. [OCA13] compared six different and popular sentiment analysis lexical resources (Harvard General Inquirer, Opinion Lexicon, Macquarie Semantic Orientation Lexicon, MPQA Subjectivity Lexicon, SentiWordNet, Emoticons) to evaluate the usefulness of each resource for stock prediction. Sprenger et al. [STSW14] used sentiment analysis on stock-related tweets collected during a six-month period. To reduce noise, they selected tweets containing cashtags (i.e. $) of S&P 100 companies. Each message was classified by a Naïve Bayes method trained with a set of 2,500 tweets. The outcome of such an analysis showed that sentiment indicators are associated with abnormal returns, and message volume is correlated to the trading volume. Similarly, Rao and Srivastava [RS14] associated a polarity to each day, considering the number of positive and negative tweets via *sentiment140*[1], and testing DJIA and NASDAQ-100 indexes in a 13-month period between 2010 and 2011. Mao et al. [MCB11b] surveyed a variety of web data sources (Twitter, news headlines and Google search queries) and tested two sentiment analysis methods for the prediction of stock market behavior. They found that their Twitter sentiment indicator and the frequency of financial terms on Twitter are statistically significant predictors of daily market returns.

Other approaches do not use sentiment analysis to make predictions. For example, Mao et al. [MWWL12] analyzed with a linear regression model the correlation between the Twitter predictor and stock indicators at three levels (i.e. stock market, sector and single company level). The authors discovered that the daily number of tweets mentioning S&P 500 stocks was

[1] http://help.sentiment140.com/

significantly correlated with S&P 500 daily closing price. Accuracy of about 68% was obtained in a 19-day test for stock market and sector level prediction, and about 52% for company stock. Porshnev et al. [PRS13] created different types of features: to a "basic" bag-of-words of the previous day tweets, they added features regarding the number of tweets containing the words *hope*, *worry* or *fear* (Basic&HWF), or the words *happy*, *loving*, *calm*, *energetic*, *fearful*, *angry*, *tired* and *sad* (Basic&8emo). Training a support vector machine model on such datasets over a 7-month period in 2013, they got a maximum baseline accuracy of 65.17% for DJIA, 57% for S&P 500 and 50.67% for NASDAQ. In a different way, Ruiz et al. [RHC$^+$12] extracted two types of features, one concerning the overall activity on Twitter and the other measuring the properties of an induced interaction graph. They found a correlation between these features and changes in S&P 500 price and volume traded. Zhang et al. [ZFG11] found high negative correlation (0.726, significant at level $p < 0.01$) between Dow Jones index and the presence of some words in tweets, such as *hope*, *fear*, and *worry*. A quantitative analysis is made by Mao et al. [MWW13]: using Twitter volume spikes in a 15-month period (from February 2012 to May 2013), they trained a bayesian classifier to assist S&P 500 stock trading, showing that it may lead to a substantial profit. Arias et al. [AAX13] showed through extensive testing that the prediction of stocks or indexes can improve by adding Twitter-related data (either in terms of volume or public sentiment) to non-linear time series (support vector machines or neural networks).

5

Recommender systems for job search

Job searching, also known as job hunting or job seeking, refers to the process people looking for a job perform in order to find it. On the other hand, finding the right employee is a key aspect for enterprises, which continuously have to recruit professionals according to their current needs. Both tasks are sides of the same general problem, namely allowing communication between companies and potential applicants for the sake of establishing an employment relationship.

Recommender Systems (RSs) are software tools and techniques that provide suggestions for items that are most likely of interest to a particular user [AT05, RRS15]. The word item refers to what is recommended, from movies to products, to jobs. The increased amount of digital information available in the age of Big Data has changed the way companies conduct their business, and also the way they employ candidates.

E-recruitment platforms spread in the recent years to address the challenge of recruiting the appropriate person. Simultaneously, this has become an opportunity for job seekers, who typically publish their profiles on job portals. Looking for a job is a tough, often tedious, task that everyone eventually has to face. Each year, millions of persons engage in job search for several reasons, including accidental job loss, come back to work, completion of job training, or the desire to pursue new career opportunities. Consequently, a large volume of job postings as well as user profiles is available online, and can be exploited to improve the matching between candidates and jobs.

Job recommender systems aim at suggesting jobs as pertinent as possible to candidates' profiles. The best fit may depend on aspects that are hard to measure, such as personal characteristics, social skills, preference for the job location, and so on. For these reasons, studies investigating job search behavior have recently proliferated, and many approaches have been proposed to match candidates with jobs.

In this chapter, an overview of recruitment systems and job recommendation systems is given, briefly describing challenges as well as research advances. Additionally, an emerging research thread is introduced, that is Career Pathway Recommendation (CPR), whose goal is to help a user achieving a career goal, possibly far into the future. The most convenient path towards the goal will be suggested to the user.

5.1 Recruitment systems

Recruitment, or recruiting, refers to the process of identifying and selecting suitable candidates for jobs within an organization. Such a process has been extensively studied in the human resources area [MWH94, AVdV01], and recruitment systems have been built accordingly [Lee07, ELW08], which are used by human resources departments to select candidates fitting the profiles enterprises are looking for.

In recent times, e-recruitment platforms have emerged to foster the recruiting process using the increased amount of web information [TBW08], especially collected from social networks, such as LinkedIn, Facebook, Twitter, Xing [ZESD14, Fle15]. These platforms have led job seekers to publish their curricula on job portals. For each posted job, thousands of resumes are received by companies; consequently, a huge volume of job descriptions and candidate resumes are becoming available online. Curricula and profiles can be exploited by recommender systems to match candidates with jobs using various approaches, including collaborative filtering, content-based filtering, knowledge-based and hybrid approaches [WHF07]. However, e-recruiting platforms are usually based on boolean search and filtering techniques that cannot sufficiently capture the complexity of a person-job fit as selection decisions [MKWW06].

Davison et al. [DMB11] pointed out that LinkedIn provides more accurate information compared to Facebook because everybody in a person's network can easily contradict her assertions. This can be among the reasons why Zide et al. [ZESD14] defined LinkedIn as the world's largest professional network. In addition to its reliability, LinkedIn also offers recruiter accounts aiming to support the recruiting process, so that about 94% of recruiters use it [Kas15]. Instead, the same trend does not hold among social media job seekers, where only 40% makes use of this network, although members are sometimes notified of possibly interesting job offers. LinkedIn professional secrecy does not allow us a complete understanding of the techniques used to recommend job positions. Anyway, analyzing some public profiles and the relative recommended job positions, we noted that there are often wrongly retrieved (i.e. not interesting) offers because of homonymy. This issue could make the job seeking process less effective, more manually conducted and time consuming.

While some works focused on the human resource aspect of the recruitment process [Bue14, PCG11], others proved the benefit of machine learning approaches for job placement. For instance, Min and Emam [ME03] used rules created by a decision tree in order to manage the recruitment of truck drivers. Buckley et al. [BMJM04] presented an automated recruitment and screening system, showing conservative savings of 6 to 1 ratio due to reduced employee turnover, reduced staffing costs and increased hiring process efficiency. Chi [Chi99] applied principal component analysis to establish jobs that can adequately be performed by various types of disabled workers. A dataset of 1,259 occupations was used, summarized in 112 different jobs; 41 available skills were analyzed with principal component analysis, finding five principal factors, namely occupational hazard, verbal communication education and training, visual acuity, body agility, and manual ability. Finally, the 112 job titles were classified into 15 homogeneous clusters, creating useful data to expand both the counselor and counselee perspectives about

job possibilities, and job requirements through the five principal factors. Zhu et al. [ZZX+16] used unsupervised learning techniques to discover recruitment market trends based on large-scale recruitment data. Specifically, their sequential latent variable model is able to capture the sequential dependencies of corporate recruitment states, and to automatically learn the latent recruitment topics.

5.2 Job recommendation systems

Since manual searches are onerous and tedious, job recommendation systems help automating job search, suggesting the most appropriate job to a given candidate [AOY12]. Researchers have identified some requirements (Figure 5.1) that should be borne in mind when recommending candidates for a given job [MKWW06, MWK08, Kei07]:

1. The matching between candidates and jobs depends on skills and abilities that individuals should have.

2. The recommendation process is bidirectional, meaning that both the preferences of the recruiter and of the candidate should be taken into account.

3. Recommendations should be based on the candidate attributes, as well as the relational aspects in the working place.

Therefore, job recommendation is a bidirectional task, where the job with the highest matching degree should be recommended to a job seeker, and the job seeker with the highest matching degree should be recommended to a job. Candidates and jobs should be ranked according to some criteria, which are supposed to affect or are related to the employee performance on the job. Malinowski et al. [MKWW06] proposed a bilateral people-job recommender system to match applicants with job opening profiles, assuming that both data are available. In order to perform the matching, a supervised probabilistic model estimates the probability that an applicant likes a job. Differently, Paparrizos et al. [PCG11] recommended job positions to applicants based only on the job history of other employees. Starting from an individual who was currently employed in an institution, they aimed at predicting the next institution where the individual would have been employed. The authors relied on preceding job positions, fields of education, educational experiences, also taking into account start and finish dates. Qin et al. [QZX+18] proposed a neural network method to link persons and jobs. Recurrent neural network was used to build a word-level semantic representation for both job requirements and job seekers' experiences. In particular, ability-aware attention strategies were designed to measure the different importance of job requirements for semantic representation, and the different contribution of each job experience to an ability requirement was also measured. Yi et al. [YAC07] introduced a structured relevance model to match resumes and jobs. Almalis et al. [ATK14] introduced a content-based approach to quantify the suitability of a candidate employee for a specific job position.

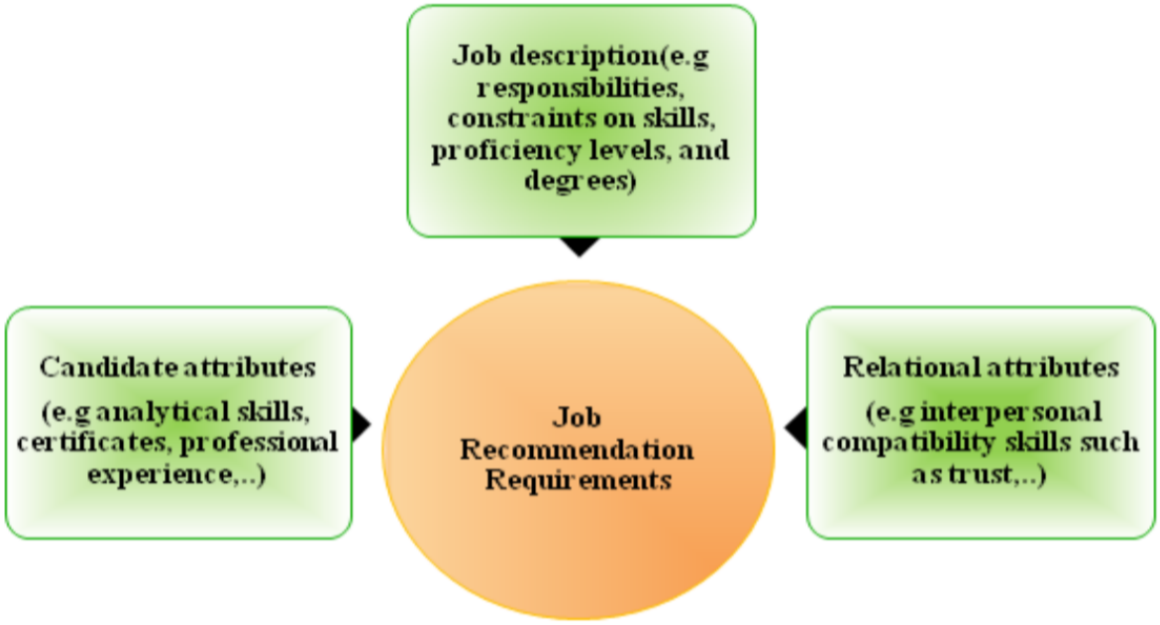


Figure 5.1: Model of system requirements for candidates/job recommendation. *Source: [AOY12].*

Other existing works [KR14, CBVS14] focused on the academic degree of students, aiming to predict both their academic performance at early stage of their curricula and their placement chance, using supervised classifiers like SVMs or neural networks. Zhang et al. [ZYN14] designed a collaborative filtering method that takes advantage of background information such as students' resumes, recruiting information, and users' applications to jobs. Rafter et al. [RBS00] introduced an online job finder engine that uses a collaborative filtering algorithm with some user preferences, and mined server logs of web recruitment platforms to get an appropriate user representation. Buettner [Bue14] introduced a recommender system based on social network information, relying on three fit measures related to candidates. The first, called organization fit, is a macro-perspective compatibility between an employee's personality and an organization's culture; the second, called group fit, focuses on a user's social interactions; the third, called job fit, is a micro-perspective matching between a candidate's abilities or preferences, and job demands. A similar goal was pursue by Gupta and Garg [GG14], who used candidate profile matching as well as preserving their job preferences to develop a job recommender.

User profiling is one of the major flaws of job recommendation approaches, because such data are hard to retrieve, select and handle. In general, the concept of candidate profiles is relevant to properly estimate a candidate's performance on jobs [SWNF12, PCG11]. Such profiles may contain:

- Personal information about the candidate, such as name, age, gender, location.
- Information about the candidate's job history, including details on the company as well as

job start and finish dates.

- Information about educational experiences, including school and university names, degrees, and possibly other details.

For the sake of building profiles, Chi [Chi99] used a set of 41 skills, falling in three major classes, that are required education and experience, physical demands, and task environment. Hong et al. [HZWS13] performed an user grouping in distinct clusters and different recommendation strategies according to users' characteristics. Other researchers [Bue14, GG14] used information related to companies, users, user preferences and social interactions. Rubin et al. [RBB02] showed the importance of extracurricular activities as a users' skills indicator. Furthermore, they pointed out that corporate recruiters and talent pool managers would benefit from a tool that allows the systematic categorization and valuing of extracurricular and contextual activities.

While the just described job recommendation approaches use features related to the current and past experiences, either in employment or education, in Chapter 10 a method that relies on a candidate's skillset will be introduced, where skillsets will be mined from LinkedIn, and used to represent both candidates and jobs. Thanks to this homogeneous representation, this approach will provide a prediction of the best job in relation to a user's capabilities and knowledge.

5.3 Career pathway recommendation systems

Career Pathway Recommendation (CPR) generalizes the idea of job recommendation. Instead of a single job only, a sequence of jobs can be recommended that identifies a career pathway either consistent with the user's job history or targeting a specific career goal. While job recommendation strives to suggest the most appropriate job for a given candidate, the purpose of CPR is to discover a pathway that leads a candidate towards her career goal, dealing with uncertainty, e.g. about alternative career choices, the likelihood that the application to a job succeeds, and so on.

Career pathway recommenders can be applied to use cases where job recommenders would inherently fail. Let us say for example that a job is exactly what a candidate is looking for, but she does not have an adequate profile to get the job. A job recommender can choose whether to recommend the job or not. If it is able to capture that the candidate's profile is not appropriate, it will not recommend the job, but the candidate will never know about a job she would have been very interested in. If it recommends the job, it will be leading the candidate towards a failure. Consequently, the candidate might lose confidence in herself, or even stop pursuing her career goal. On the other hand, a career pathway recommender will recommend a policy that eventually leads the user towards her goal, minimizing the intermediate career steps while simultaneously improving the user's profile and increasing the likelihood that the user succeeds when applying for her career goal. In other words, a career pathway recommender will take the career goal into account, and will help the user finding a pathway to it rather than recommending it when the user's application is very likely to fail.

Despite the fact that many approaches have been proposed for job recommendation, little effort has been made to the problem of recommending a pathway that leads users towards a career goal. Elayidom et al. [EIA11] proposed a decision tree based approach which helps students choosing a good branch that may fetch them placement in either rural or urban sectors. In other works [ASE14, RHN+14], the epithet career pathway recommendation is used with a different meaning. The authors employed it alluding to a branch of potentially interesting jobs rather than to a sequence of jobs aiming at achieving a final goal. Mimno and McCallum [MM08] analyzed the text of a corpus of resumes and their job transitions to mine latent skills that make up each job description and modeling career path trajectories. Differently from their work, where the past job history of all users is used to learn general path trajectories, the approach that will be presented in Chapter 11 provides a personalized recommendation for each user based on her profile.

On the other hand, some researchers focused on factors that can affect talent career steps. Li et al. [LGZ+17] analyzed two critical issues in talent management, namely turnover and career progression, showing that they can effectively be predicted through a survival analysis, which considers sequences of time intervals. Xu et al. [XYY+16] created a job transition network where vertices stand for organizations and a directed edge represents the talent flow between two organizations for a time period. Talent circles including the organizations with similar talent exchange patterns may be detected from such a network, and their characteristics can be used for talent recruitment and job recommendation. Xu et al. [XYX+15] tried to discover the job change motivations as well as correlations between professional and daily life. The authors used the career mobility and daily activity patterns at the individual level to assess to what extent such data can be predictors of a job change. Xu et al. [XZZ+17] introduced a data driven approach for modeling the popularity of skills based on the analysis of large-scale recruitment data. After having used a large corpus of job postings to build a job skill network, they created a topic model that integrates different criteria of jobs (e.g. salary, company size) and the latent connections within skills to effectively rank the skills based on their popularity.

The career pathway recommendation problem can be formulated as a Markov Decision Process (MDP) [Put14]. MDP-based systems have already been proved useful for product recommendation [SHB05], where the goal is recommending a product given any sequence of user purchases. In career pathway recommendation the goal is somewhat related, and the MDP allows extracting an optimal sequence of jobs (i.e. policy) considering every possible career evolution. Differently from product recommendation, where the recommendation of an item increases its probability to be bought, the recommendation of a job does not affect the probability for a user to get it.

6 Big Data Analytics

This chapter focuses on the analytic process, putting the emphasis on what is needed to properly perform an analysis. As the reader might expect, the focus is not only on the algorithms, but also on Big Data mining platforms. In particular, a description of the main characteristics of such platforms is provided, such as computational resources, storage, together with advantages and drawbacks.

The rationale is to provide readers with some background knowledge that helps understanding the possibilities that big data offer along with the challenges that programmers have to face when conducting an analysis. Specifically, the large variety of platforms available makes it hard to decide which solution fits best the current requirements. Because of this awkward choice and the continuously changing requirements, porting algorithms to different platforms is often needed.

In order to reduce the development time of algorithms, which typically need to be rewritten from scratch, parallel programming primitives for platform-independent algorithm implementation will be introduced in Part V.

6.1 Introduction

In the age of Big Data, the traditional data analytics may not be able to handle such large quantities of data. The data are not only large, but also heterogeneous, composed of various data types, and even including streaming data [R+11]. They come from sensors, devices, third parties, Web applications, and social media, and they are typically delivered at various speeds and frequencies.

In order to analyze such big and heterogeneous data, we can rely on advanced analytic tools based on predictive analytics, data mining, statistics, artificial intelligence, natural language processing, and so on. Big data analytics consists in the application of advanced analytic techniques to very big data sets.

Big data analytics is a compelling business opportunity. Through exploratory and detailed analysis of big data, an organization can get insights on their customers, markets, products, operations, which can be used for business advantage. However, dealing with big heterogeneous data and advanced analytic techniques is not trivial. In particular, two main problems should be tackled in order to support the analytic process:

- High performance platforms should be developed to efficiently conduct a data analysis.
- Suitable mining algorithms should be designed to get useful insights from large-scale data.

The rest of this chapter describes the characteristics of Big Data platforms as well as the enhancements in Big Data mining algorithms.

6.2 Platforms

High performance hardware platforms are becoming increasingly important for the analysis of big data, and choosing the right hardware/software platforms is crucial if the user's requirements have to be satisfied in a reasonable amount of time [SR15]. Since many platforms exist with different characteristics, the choice of the most suitable platform requires an in-depth knowledge of all viable alternatives.

The essential feature every platform incorporates is scalability, namely the capability to grow and manage an increased demand in a reasonable amount of time. The big data platforms can be divided according to the following two types of scaling:

Horizontal scaling (scale out): the workload is distributed across many independent machines to improve the processing capability. Multiple instances of the operating system are typically running on separate machines.

Vertical scaling (scale up): involves installing new hardware typically within a single server, e.g. more processors, more memory and faster hardware. It usually involves a single instance of an operating system.

Horizontal scaling increases the performance in small steps, reducing the financial investment required to upgrade. Moreover, the system can scale out as much as needed, without any theoretical limitation. On the other hand, software has to handle the data distribution and has to deal with parallelism. Few software frameworks are available that can take advantage of horizontal scaling.

Vertical scaling is very handy to set up, because managing and installing hardware within a single machine is straightforward. Differently from horizontal scaling, most of the software can easily take advantage of vertical scaling. However, a substantial financial investment is required to scale up a machine. To handle future workloads, the user typically invests more than what is required for the current processing needs. This is due to the limited space and the number of expansion slots available in a single machine, which limit vertical scaling.

6.2.1 Horizontal scaling platforms

The most known platforms based on horizontal scaling are peer-to-peer networks [MKL+02, SW05], Apache Hadoop [Had09] and Spark [ZCF+10].

Peer-to-peer networks

Peer-to-peer networks are a decentralized and distributed network architecture where the nodes in the network (i.e. peers) serve and consume resources, typically through message passing. While broadcasting messages is cheap, the aggregation of data and results is much more expensive.

The standard communication paradigm of peer-to-peer networks is Message Passing Interface (MPI), which has several desirable features: the ability to preserve the state of the computation; the hierarchical master/slave paradigm to support dynamic resource allocation where the slaves have large amounts of data to process; the barrier mechanism to synchronize processes at a certain point of the computation. On the other hand, MPI does not include a fault tolerance mechanism, so that a single node failure in a peer-to-peer network can cause the entire system to shut down. Since the enforcement of fault tolerance is completely transferred to the user, MPI is no longer widely used.

Hadoop

Hadoop is an open source framework for storing and processing large datasets using clusters of commodity hardware. Hadoop can scale up to thousands of nodes, and it is also fault tolerant, differently from MPI and peer-to-peer networks. The Hadoop platform contains various components (Figure 6.1), whose most important are a distributed file system (HDFS) [B+08] and a resource management layer (YARN) [VMD+13]. The former is used to store data across cluster of machines while providing high availability and fault tolerance, whereas the latter schedules the jobs across the cluster.

Figure 6.1: Hadoop Stack showing different components. *Source: [SR15].*

The programming model used in Hadoop is MapReduce [DG08], according to which a task is broken into two parts, namely mappers and reducers. The former read the data from HDFS, process them, and generate intermediate results, which are then aggregated by the latter to generate the final output to be written to HDFS.

Some wrappers have been developed for MapReduce to foster the source code development, including Apache Pig by Yahoo! [ORS+08] and Hive by Facebook [TSJ+09]. One of the major

drawbacks of MapReduce is its inefficiency in running iterative algorithms, because MapReduce is not designed for iterative processes. A few approaches have been developed to circumvent this limitation [BHBE10, EPF08, PR12], which however introduce additional levels of complexity in the source code.

Spark

Spark is a next generation paradigm for big data processing developed by researchers at the University of California in Berkeley. Spark is able to perform in-memory computations, so as to allow data to be cached in memory, eliminating the Hadoop's disk overhead limitation for iterative tasks. Generally, it is up to 100x faster than Hadoop MapReduce when data fit in memory, and up to 10x faster when data reside on disk. Spark can run on Hadoop Yarn manager, use HDFS, and support Java, Scala and Python, making it a general and versatile engine for large-scale data processing that may run on different systems.

6.2.2 Vertical scaling platforms

The most famous vertical scaling platforms include High Performance Computing Clusters (HPC) [B$^+$99], Field Programmable Gate Arrays (FPGA) [BFRV12], Multicore processors [BBL11] and Graphics Processing Unit (GPU) [OHL$^+$08].

HPC clusters

HPC clusters, sometimes referred as supercomputers, are machines with thousands of cores. Depending on requirements, they can have a different kind of disk organization, cache, communication mechanism, and so on. HPC clusters typically use MPI as the communication scheme, but fault tolerance is not a big deal here due to the top quality hardware employed, which makes failures extremely rare. Such top quality hardware is optimized for speed and throughput; as a consequence, the initial cost of deployment is very high. HPC clusters are not as scalable as Hadoop or Spark clusters, because the cost of scaling up such a system is much higher, but they are still able to process terabytes of data.

FPGAs

FPGAs are specialized hardware units that are custom-built for specific applications, and can be highly optimized for speed, e.g. orders of magnitude faster than other platforms. They are programmed using Hardware Descriptive Language (HDL) [TM08] at a higher development cost because of the customized hardware. In spite of being useful for a variety of real world applications, in particular network security [CCS11], the speed of multicore processors is recently reaching closer to that of FPGAs.

Multicore processors

Multicore processors refer to one machine having dozens of processing cores, which typically have shared memory but only one disk. Recently, CPUs have gained internal parallelism, the number of cores per chip and the number of operations that a core can perform have significantly increased, and multiple CPUs are allowed within a single machine, supporting algorithm acceleration. The parallelism in CPUs is mainly achieved through multithreading, where a task is broken down into threads, which are executed in parallel on different CPU cores, sharing the same memory. A high-level support to multithreading is provided by most of the programming languages through libraries.

GPUs

Graphics Processing Unit (GPU) is a specialized hardware designed to accelerate the creation of images in a frame buffer intended for display output. Although GPUs were originally used just for video and image editing, general-purpose computing on graphics processing units (GPGPU) [ND10] has recently arisen.

A GPU has large number of processing cores, e.g. 3,000+, and has its own high throughput DDR5 memory, which is many times faster than a DDR3 memory. A GPU usually has two levels of parallelism: it contains multiprocessors (MPs) and within each multiprocessor there are several streaming processors (SPs). A GPU-based program breaks down into threads that execute on SPs, then these threads are grouped to form blocks that run on a MP. Each thread within a block can only communicate and synchronize with other threads in the same block, accessing a small portion of shared cache memory and a larger portion of global main memory. Since inter-block communication is not allowed, any task needs to be broken into blocks that can run independently of others.

Nvidia has recently launched Tesla series of GPUs for high performance computing, and the CUDA framework to make GPU programming accessible to software programmers. Consequently, GPUs have been used in the development of faster machine learning algorithms based on the CUDA framework. Beyond the advantages, GPUs also have drawbacks. First, they have a limited memory, not suitable to handle terabytes of data; when data exceed the size of memory, the disk access becomes a bottleneck. Second, few algorithms are easily portable to GPUs, because of the way in which the task is required to be broken into blocks.

CPUs vs GPUs

The processing power of GPUs is much higher than CPUs (i.e. 3,000+ processing cores with 1,000Tflops of processing power for a GPU, compared with tens of processing cores with 10Gflops of processing power for a CPU). The main drawbacks of CPUs are their limited number of processing cores and their dependence on the system memory for data access, which is the major bottleneck. GPU avoids this by using a DDR5 memory, much faster than a DDR3 memory

used in a system. Moreover, in order to accelerate the data access, GPU uses a high speed cache for each multiprocessor.

6.3 Algorithms

Along with high performance platforms that support the data analytic process, suitable mining algorithms are also required to effectively retrieve valuable information from big data. In this section, a brief overview is given of the main challenges involving the creation of Big Data mining algorithms. The main challenges concerning data mining are represented by the big data volumes, distributed data distributions, and by complex and dynamic data characteristics [WZWD14].

In particular, distributed, heterogeneous, sparse, uncertain, incomplete, and multi-source data are pre-processed and blended into a unique, consistent data source. After pre-processing, complex and dynamic data are mined to get insights, which are fed back to the pre-processing stage and are useful to refine the model and parameters. Information sharing is essential throughout the process.

Mining methods have been adapted to Big Data scenarios in several ways, for example introducing parallel implementations of traditional knowledge discovery and data mining algorithms [CBZ09], designing data mining mechanisms from a multi-source perspective [WZ03], and proposing frameworks for mining high-speed data streams [DH00], which cannot effectively be analyzed by means of static knowledge discovery methods.

Discovering knowledge from massive data typically entails heterogeneous data sources, where data may come as streams. Differently from single-source mining methods, a multi-source data mining mechanism should take into account all these characteristics of data coming from multiple sources, that are large-scale, heterogeneous, and real-time.

Wu et al. [WZ03] proposed the theory of local pattern analysis, which is an alternative to aggregating data into a single source, usually infeasible in practice due to transmission costs and privacy concerns. Knowledge evolution is a common phenomenon in real world systems, that are characterized by frequent changings. In the knowledge discovery process, concept drifting aims at analyzing target concept changes as well as context changes in data streams. According to different types of concept drifts, knowledge evolution can take forms of mutation drift, progressive drift, and data distribution drift, based on single features, multiple features, and streaming features [WYD+13].

6.3.1 Local learning and model fusion for multiple information sources

Big Data applications typically involve many autonomous sources with decentralized controls. A Big Data mining system should not consider these autonomous sources separately to avoid ending up with partial and biased models. Although the aggregation of such data sources would be desirable, it is infeasible in practice due to transmission costs and privacy concerns. A partial

solution to this issue is to allow information exchange among different data sources in order to achieve a global goal.

The global mining goal can be reached with a two-step process, including local mining and global correlation, at data, model, and knowledge levels. At the data level, each autonomous source can compute statistics based on local data, then share the result with the other sources in order to achieve a global view of data distribution. At the model level, each autonomous source can analyze local data to discover local patterns, whose sharing with other sources can bring to the synthesis of new global patterns [WZ03]. At the knowledge level, correlation between the different local models is analyzed, so as to discover the mutual relevance between the data sources and to create robust global decision models.

6.3.2 Mining from sparse, uncertain and incomplete data

Data sparsity is one of the major problems that affect data processing and understanding. Generally, it is a consequence of data dimensionality, where a high number of dimensions may bring data to have an unclear distribution. Since most mining algorithms are affected by the so called "curse of dimensionality" [KM11], dimensionality reduction techniques, e.g. feature selection, feature transformation, are used to mitigate the problem.

When data are affected by errors or some source of randomness, we talk about uncertain data. Data produced by physical devices, for instance GPS data, are inherently uncertain, because the precision is limited due to technological constraints. The major challenge is that each object is represented as an interval rather than as a single value, so most existing data mining algorithms cannot directly be applied. Common solutions involve the estimation of model parameters taking the data distributions into consideration [WZ08].

Incomplete data refers to data field values that can be missing because of the malfunction of a sensor, a policy that allows to skip some values, some privacy constraints, and so on. Data mining algorithms usually handle missing values by ignoring such fields, but improved models could be produced by replacing missing values with more significant data. For this purpose, the major approaches consist in filling entries with the most frequently observed values, or building learning models to predict possible values for each data field, based on the observed values of a given instance [Efr94].

6.3.3 Mining complex and dynamic data

The growth of Big Data is driven by the increase in volume of complex data, such as Web documents, social networks, communication networks [Bir12]. Complex data offer compelling opportunities, as witnessed for example by the considerable amount of researches that used Twitter for sentiment analysis and stock market prediction, as discussed in Section 4.4.

Making use of complex data is challenging due to scalability issues, because each node in a network is potentially connected with all the others. Several researchers have already faced this

problem, for example to find communities and trace their dynamically evolving relationships [Cen10], or to discover outliers in social networks [BMBL09].

Big Data complexity can be seen in different aspects, such as heterogeneous data types, intrinsic semantic associations in data, and relationship networks among data. Data types are extremely heterogeneous: apart from the traditional relational databases, also texts, hypertexts, images, audios, videos, etc. emerge and spread out in the age of Big Data. Traditional data models, such as key-value stores, big tables, document databases, and graph databases, are not able to handle complexity in an efficient as well as effective way. Intrinsic semantic associations can typically be found in texts, images, videos on the Web, whose discovery may help to improve the performance of systems like search engines or recommendation systems. However, describing semantic features and building association models are challenging tasks due to the heterogeneity of big data sources. Finally, examples of relationship networks are the Web, where the pages linking to each other via hyperlinks form a complex network, the social relationships between persons in social networks, the sensor networks, and so on. To deal with complex relationship networks, emerging research efforts have begun to address the issues of structure-and-evolution, crowds-and-interaction, and information-and-communication [WZWD14].

Part II

Algorithms and Methods for Sentiment Analysis

7

Markov techniques for transfer learning and sentiment analysis

This chapter introduces novel methods to identify the sentiment orientation of text documents, possibly reusing models learned on different domains. In particular, such approaches are able to discover the polarity of a plain text in traditional in-domain contexts, and can automatically handle knowledge transfer across heterogeneous domains, supporting model reuse.

All the proposed techniques rely on strong mathematical foundations, since they are based on the well-known Markov chain theory, which is applied in a wide range of topics, such as computer science, speech recognition, game theory, sports, physics, medicine, and so on.

7.1 Markov chain

In order for readers to properly understand the remaining of this chapter, a very short introduction to Markov chains is given, briefly describing their working principles as well as their applications.

7.1.1 Introduction

A Markov process is a stochastic process that satisfies the Markov property, namely each state of the process only depends on the previous one, whereas it is statistically independent of all other states [Ser09]. A Markov chain is a type of Markov process that is characterized by a discrete state space. More precisely, a Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event [Gag17]. This property is sometimes referred as memorylessness. Markov chains can be represented by graphs, whose nodes represent the discrete state space and whose transitions represent the probability to move from one state to another.

For example, in Figure 7.1 the transition probabilities between different weather conditions are modeled as a Markov chain, whose states are *Cloudy*, *Sunny* and *Rainy*. According to the figure, if weather is *Cloudy* there are a probability of 0.4 that it will become *Sunny*, a probability of 0.4 that it will become *Rainy*, and a probability of 0.2 that it will stay *Cloudy*. More formally, $P(Sunny|Cloudy) = 0.4$, $P(Rainy|Cloudy) = 0.4$, $P(Cloudy|Cloudy) = 0.2$. As can be noted,

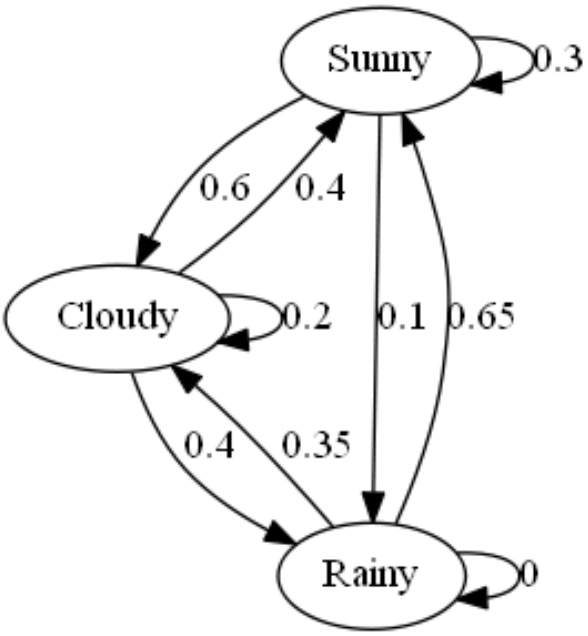


Figure 7.1: Example of transition probabilities between different weather conditions.

there is no evidence of how much time is required for transitions to occur. Time can be defined as either discrete or continuous, but the most relevant aspect is to establish the transition probability between subsequent time steps.

Careful readers can find further details and properties of Markov chains in [Lam12].

7.1.2 Markov theory applications

Markov chain theory has been successfully applied in several text mining contexts, such as information retrieval, sentiment analysis, text classification.

Markov chains are particularly suitable for modeling hypertexts, which can be seen as graphs where pages or paragraphs represent states and links represent state transitions. Such an approach may help in information retrieval tasks, because it allows discovering the possible presence of patterns when humans search for information in hypertexts [Qiu93], performing link prediction and path analysis [Sar00] or, even, defining a ranking of Web pages just dealing with their hypertext structure, regardless of information about page content [PBMW99].

Markov chains, in particular hidden Markov chains, have also been employed to build information retrieval systems where query, document or both are expanded, then the most relevant documents with respect to a given query are retrieved [MS94, MLS99], possibly in a spoken document retrieval context [PLL12] or in the cross-lingual area [XW00]. Anyhow, to fulfil these purposes, Markov chains are exploited to model term relationships. Specifically, they are used either in a single-stage or in a multi-stage fashion, the latter just in case indirect word

relationships need to be modeled [CNB07].

The idea of modeling word dependencies by means of Markov chains is also applied in sentiment analysis. Indeed, hidden Markov models (HMMs) find out opinion words (i.e. words expressing sentiment) [LHZ10], possibly trying to correlate them with particular topics [MLW+07, JO11]. Typically, transition probabilities and output probabilities between states are estimated by using the Baum-Welch algorithm, whereas the most likely sequence of topics and related sentiments is computed through the Viterbi algorithm. The latter also fosters Part-of-speech (POS) tagging, where Markov chain states not only model terms but also tags [JHS09, NY03]. In fact, when a tagging is demanded for a sequence of words, the goal is to find the most likely sequence of tags for that sequence of words.

Following works focus on text classification, where a widespread approach based on Markov models consists in building a HMM for each different category. The idea is, for each given document, to evaluate the probability of being generated by each HMM, finally assigning to that document the class corresponding to the HMM maximizing such a probability [YB08, XSH+06, VID, YB13]. Beyond directly using HMMs to perform text categorization, they can also be exploited to model inter-cluster associations. For instance, words in documents can be clustered for dimensionality reduction purposes, and each cluster can be mapped to a different Markov chain state [LD13]. Another interesting application is the classification of multi-page documents where, modeling each page as a different bag-of-words, a HMM can be exploited to mine correlation between the documents to be classified (i.e. pages), by linking concepts in different pages [FSV02].

7.2 A Markov method for cross-domain sentiment classification

This section introduces a novel method inspired by Markov chain theory, specifically designed for cross-domain sentiment classification, but also suitable for in-domain tasks. Such a method offers a mechanism to accomplish transfer learning from a source to a target domain, dealing with the heterogeneity of language and paving the way for effective model reusability.

7.2.1 Overview

The basic idea is to model semantic information looking at term distribution within a corpus. To accomplish this task, the corpus of documents is represented as a graph characterized by a node for each term and a link between each pair of co-occurring terms. In such a way, a semantic network is molded, where information flows allowing transfer learning from source specific terms to target specific ones. Similarly, categories are added to graph nodes in order to model semantic information between source terms and classes.

The semantic graph is implemented by using a Markov chain, whose states represent both terms and classes, modeling state transitions as functions of the term(-class) co-occurrences in

documents. The terms in Markov chain belong to either the source domain or the target domain, whereas connections with class states are available only for source terms. This mathematical model is appropriate to spread semantic information throughout the network by performing steps (i.e. state transitions) within the Markov chain. The simultaneous existence of terms belonging to the source domain, terms belonging to the target domain, and categories, allows both transfer learning and sentiment classification. The proposed approach can be split into three main stages, namely, the text pre-processing phase, the learning phase and the classification phase. The last two phases are the most innovative parts, since they accomplish both transfer learning and sentiment classification by means of only one abstraction, that is, Markov chain.

For the first time, a Markov chain is used as a sentiment classifier rather than being just employed in a support task as part-of-speech (POS) tagging. Moreover, considering categories as Markov chain states is a novelty with reference to text classification, where the most common approach consists in building different Markov chains (e.g. one for each category) and evaluating the probability that a test document is being generated by each of them.

7.2.2 Text pre-processing

The first stage of the algorithm is text pre-processing. Starting from a corpus of documents written in natural language, the goal is to transform it in a more manageable, structured format.

Initially, standard techniques are applied to the plain text, such as word tokenization, punctuation removal, number removal, case folding, stopwords removal and the Porter stemming algorithm [Por80]. It should be noted that stemming supports sentiment classification, because words with the same morphological root are likely to be semantically similar.

The representation used for documents is the common bag-of-words, that is, a term-document matrix where each document d is seen as a multiset (i.e. bag) of words (or terms). Let $\mathscr{T} = \{t_1, t_2, ..., t_k\}$, where k is the cardinality of $\mathscr{T}$, be the dictionary of terms, which typically includes all terms in the corpus to be analyzed. In each document d, each word t is associated to a weight w_t^d, usually independent of its position inside d. More precisely, w_t^d only depends on term frequency $f(t,d)$, that is, the number of occurrences of t in document d, and in particular, represents relative frequency $rf(t,d)$, which is computed as follows:

$$w_t^d = rf(t,d) = \frac{f(t,d)}{\sum_{\tau \in \mathscr{T}} f(\tau,d)} \tag{7.1}$$

After having built the bags of words, a feature selection process is performed to tackle the curse of dimensionality. In fact, the larger is the dataset to be analyzed the higher k tends to be. Feature selection allows retaining only the most promising terms for the classification process. Moreover, choosing only a small subset of terms cuts down the computational burden required by the learning and classification phases.

Feature selection is essential, and may considerably affect the effectiveness of the whole method. Thus, it is critical to pick out a feature set as good as possible to support the classification

process. For this purpose, some feature selection approaches, which will be presented below, are experimented.

Feature selection variants

The first feature selection method employed for experimentation is based on document frequency, $df(t)$, defined as:

$$df(t) = |\{d : t \in d\}| \tag{7.2}$$

Each term with df higher than an established threshold is selected.

The second alternative consists in selecting only the terms included in a list (i.e. an opinion wordlist) of commonly used words to express sentiment, regardless of their frequency, their document frequency or other aspects. Such words are just terms with well-known polarities. The opinion wordlist used here has been proposed in [Liu12]. It contains 2,003 positive words and 4,780 negative words in plain English.

The previously presented feature selection methods are both unsupervised. Another viable option that exploits the knowledge of class labels is described straight away. First of all, a supervised scoring function is used in order to find the most relevant features with respect to their ability to characterize the categories. Such a function is either information gain (IG) or chi-square (χ^2), defined as in [DMPS15]. The ranking obtained as output is used to select the best n features as well as to change term weighting within documents. In fact, this score $s(t)$ is a global value, stating the relevance of a certain word t, whereas relative frequency, introduced by Equation (7.1), is a local value that only measures the relevance of a word within a particular document. Therefore, these values can be combined into a novel, different term weighting to be used for the bag-of-words representation, so that the weight w_t^d comes to be

$$w_t^d = rf(t,d) \cdot s(t) \tag{7.3}$$

Thus, according to Equation (7.3), both factors (i.e. the global relevance and the local relevance) are taken into account.

7.2.3 Learning phase

The learning phase is the second stage of the method. As in any categorization problem, the primary goal is to learn a model from a training set, so that a test set can be classified accordingly. Though, the mechanism should also allow transfer learning in cross-domain tasks.

The basic idea consists in modeling term co-occurrences: the more words co-occur in documents the more their connection should be stronger. This scenario can be represented as a graph whose nodes are words and whose edges represent the strength of their connections. Considering a corpus of documents $\mathscr{D} = \{d_1, d_2, ..., d_N\}$ and a dictionary $\mathscr{T} = \{t_1, t_2, ..., t_k\}$,

$A = \{a_{ij}\}$ is the set of connection weights between the term t_i and the term t_j. Each a_{ij} can be computed as follows:

$$a_{ij} = a_{ji} = \sum_{d=1}^{N} w_{t_i}^d \cdot w_{t_j}^d \tag{7.4}$$

The same strategy could be followed to find the polarity of a given word, unless having an external knowledge base which states that a word is intrinsically positive, negative or neutral. Co-occurrences between words and classes are modeled for each document whose polarity is given. Again, a graph whose nodes are either terms or classes and whose edges represent the strength of their connections is suitable to represent such relationships. In particular, given that $\mathscr{C} = \{c_1, c_2, ..., c_M\}$ is the set of categories and $B = \{b_{ij}\}$ is the set of edges between a term t_i and a class c_j, the relationship between a term t_i and a class c_j is strengthened if t_i occurs in documents belonging to the set $D^j = \{d \in \mathscr{D} : c_d = c_j\}$.

$$b_{ij} = \sum_{d \in D^j} w_{t_i}^d \tag{7.5}$$

Careful readers may have noted that the graph representing both term co-occurrences and term-class co-occurrences can easily be interpreted as a Markov chain. Graph vertices can simply be mapped to Markov chain nodes and graph edges can be split into two directed edges, i.e. the edge linking states t_i and t_j is split into one directed edge from t_i to t_j and another directed edge from t_j to t_i. Moreover, for each state a normalization step of all outgoing arcs is enough to satisfy the probability unitarity property. Finally, Markov property holds because each state only depends on directly linked states, since co-occurrences of just two terms (or a term and a class) at a time are taken into account.

Now that the basic idea behind the method has been introduced, readers can figure out how the learning phase is performed. A subset of common terms between source and target domains is supposed to exist, which acts as a bridge between domain specific terms, allowing and supporting transfer learning. Such common terms are the key to let information about classes flow from source specific terms to target specific terms, exploiting term co-occurrences, as shown in Figure 7.2.

It should be pointed out that the just described transfer learning process is not an additional step to be added in cross-domain problems; on the contrary, it is implicit in the Markov chain mechanism, and it is also performed for in-domain tasks. Obviously, if both the training set and the test set are from the same domain, it is likely that most terms in the test documents already have a polarity.

Apart from transfer learning, the proposed Markov chain method also fulfils the primary goal of the learning phase, that is, building a model that can be subsequently used for the classification phase. Markov chain can be represented as a transition matrix (MCTM), composed of four logically distinct submatrices, as shown in Table 7.1. It is a $(k+M) \times (k+M)$ matrix, having current states as rows and future states as columns. Each entry represents a transition probability,

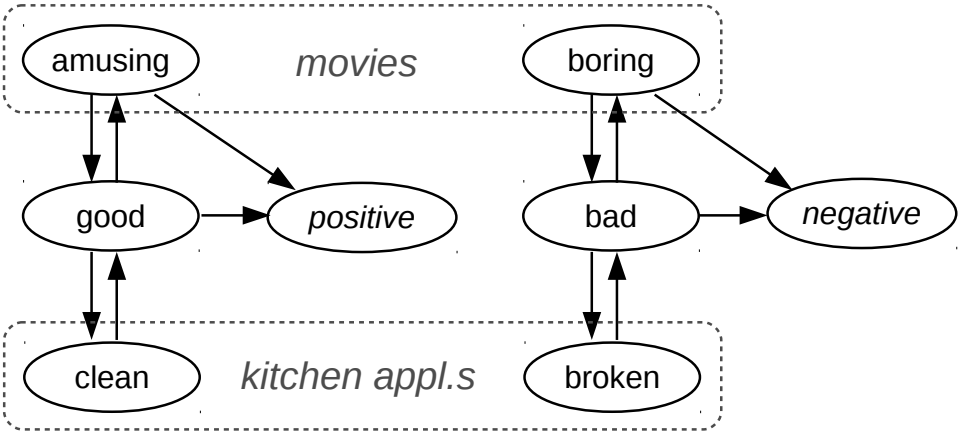


Figure 7.2: Transfer learning from movie-specific terms to kitchen appliance-specific terms through common terms. *Source: [DMPP].*

which is computed differently depending on the type of current and future states (term or class), as will be described below.

	$\mathbf{t_1}, ..., \mathbf{t_k}$	$\mathbf{c_1}, ..., \mathbf{c_M}$
$\mathbf{t_1}, ..., \mathbf{t_k}$	$A^{'}$	$B^{'}$
$\mathbf{c_1}, ..., \mathbf{c_M}$	E	F

Table 7.1: The Markov chain transition matrix (MCTM) is composed of four submatrices, representing the transition probabilities that, starting from a current state (i.e. row), a future state (i.e. column) is reached. Both current states and future states can be either terms or classes.

Let $\mathscr{D}_{train}$ and $\mathscr{D}_{test}$ be the subsets of corpus $\mathscr{D}$ chosen as the training set and the test set respectively. The set A, whose entries are defined by Equation (7.4), can be rewritten as

$$a_{ij} = a_{ji} = \begin{cases} 0, & i = j \\ \sum\limits_{d \in \mathscr{D}_{train} \cup \mathscr{D}_{test}} w_{t_i}^d \cdot w_{t_j}^d, & i \neq j \end{cases} \tag{7.6}$$

and the set B, whose entries are defined by Equation (7.5), can be rewritten as

$$b_{ij} = \sum_{d \in D_{train}^j} w_{t_i}^d \tag{7.7}$$

where $D_{train}^j = \{d \in \mathscr{D}_{train} : c_d = c_j\}$. The submatrices $A^{'}$ and $B^{'}$ are the normalized forms of Equation (7.6) and Equation (7.7), computed so that each row of Markov chain satisfies the

probability unitarity property. Instead, each entry of the submatrices E and F looks like as follows:

$$e_{ij} = 0 \tag{7.8}$$

$$f_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases} \tag{7.9}$$

Note that E and F deal with the assumption that classes are absorbing states, which can never be left once reached.

7.2.4 Classification phase

The last step of the algorithm is the classification phase. The aim is classifying the test set by using the model learned in the previous step. According to the bag-of-words representation, a document $d_t \in \mathscr{D}_{test}$ to be classified can be expressed as follows:

$$d_t = (w_{t_1}^{d_t}, ..., w_{t_k}^{d_t}, c_1, ..., c_M) \tag{7.10}$$

$w_{t_1}^{d_t}, ..., w_{t_k}^{d_t}$ is the probability distribution representing the initial state of the Markov chain transition matrix, whereas $c_1, ..., c_M$ are trivially set to 0. The initial hypothesis is to be in many different states (i.e. every state t_i so that $w_{t_i}^{d_t} > 0$) at the same time. Then, simulating a step inside the Markov chain transition matrix, a posterior probability distribution is obtained not only over terms, but also over classes. In such a way, estimating the posterior probability that d_t belongs to a certain class c_i, the most likely label $c_i \in \mathscr{C}$ could be assigned to d_t. The posterior probability distribution after one step in the transition matrix, starting from document d_t, is:

$$d_t^* = (w_{t_1}^{d_t^*}, ..., w_{t_k}^{d_t^*}, c_1^*, ..., c_M^*) = d_t \times MCTM \tag{7.11}$$

where d_t is a column vector having size $(k+M)$ and $MCTM$ is the Markov chain transition matrix, whose size is $(k+M) \times (k+M)$. The category that will be assigned to d_t is computed as follows:

$$c_{d_t} = \arg\max_{i \in C^*} c_i^* \tag{7.12}$$

where $C^* = \{c_1^*, ..., c_M^*\}$ is the posterior probability distribution over classes.

7.2.5 Computational complexity

The computational complexity of the method is determined by the learning phase and the classification phase. Regarding the learning phase, the computational complexity overlaps with the time needed to build the Markov chain transition matrix, say $time(MCTM)$, which is

$$\begin{aligned} time(MCTM) = time(A) + time(B) + \\ + time(A^{'} + B^{'}) + time(E) + time(F) \end{aligned} \tag{7.13}$$

Remember that A and B are the submatrices representing the state transitions having a term as the current state. Similarly, E and F are the submatrices representing the state transitions having a class as the current state. The expression $time(A^{'} + B^{'})$ denotes the temporal length of the normalization step, necessary in order to observe the probability unitarity property. On the other hand, E and F are simply a null and an identity matrix, requiring no computation. Thus, since time complexity depends on these factors, all should be estimated.

The only assumption we can do is that in general $|\mathscr{T}| >> |\mathscr{C}|$. The time needed to compute A is $O(\frac{|\mathscr{T}|^2}{2} \cdot (|\mathscr{D}_{train}| + |\mathscr{D}_{test}|))$, which in turn is equal to $O(|\mathscr{T}|^2 \cdot (|\mathscr{D}_{train}| + |\mathscr{D}_{test}|))$. Regarding transitions from terms to classes, building the submatrix B requires $O(|\mathscr{T}| \cdot |\mathscr{C}| \cdot |\mathscr{D}_{train}|)$ time. In sentiment classification problems, it can also be assumed that $|\mathscr{D}| >> |\mathscr{C}|$ and, as a consequence, the previous time becomes $O(|\mathscr{T}| \cdot |\mathscr{D}_{train}|)$. The normalization step, which has to be computed one time only for both A and B, is $O(|\mathscr{T}| \cdot (|\mathscr{T}| + |\mathscr{C}|) + |\mathscr{T}| + |\mathscr{C}|) = O((|\mathscr{T}| + 1) \cdot (|\mathscr{T}| + |\mathscr{C}|))$, which can be written as $O(|\mathscr{T}|^2)$ given that $|\mathscr{T}| >> |\mathscr{C}|$. Furthermore, building the submatrix E requires $O(|\mathscr{T}|^2)$ time, whereas for submatrix F $O(|\mathscr{T}| \cdot |\mathscr{C}|)$ time is needed, which again can be written as $O(|\mathscr{T}|)$ given that $|\mathscr{T}| >> |\mathscr{C}|$. Therefore, the overall complexity of the learning phase is

$$\begin{aligned} time(MCTM) \simeq time(A) = \\ = O(|\mathscr{T}|^2 \cdot (|\mathscr{D}_{train}| + |\mathscr{D}_{test}|)) \end{aligned} \tag{7.14}$$

In the classification phase, two operations are performed for each document to be categorized: the matrix product in Equation (7.11), which requires $time(MatProd)$, and the maximum computation in Equation (7.12), which requires $time(Max)$. Hence, as can be seen below

$$time(CLASS) = time(MatProd) + time(Max) \tag{7.15}$$

the classification phase requires a time that depends on the previous mentioned factors. The matrix product can be computed in $O((|\mathscr{T}| + |\mathscr{C}|)^2 \cdot |\mathscr{D}_{test}|)$ time, which can be written as $O(|\mathscr{T}|^2 \cdot |\mathscr{D}_{test}|)$ given that $|\mathscr{T}| >> |\mathscr{C}|$. On the other hand, the maximum requires $O(|\mathscr{C}| \cdot |\mathscr{D}_{test}|)$ time. Since the assumption that $|\mathscr{T}| >> |\mathscr{C}|$ still holds, the complexity of the classification phase can be approximated by the calculus of the matrix product.

Lastly, the overall complexity of the algorithm, say $time(Algorithm)$, is as follows:

$$\begin{aligned} time(Algorithm) &= time(MCTM) + time(CLASS) \simeq \\ &\simeq time(MCTM) = O(|\mathscr{T}|^2 \cdot (|\mathscr{D}_{train}| + |\mathscr{D}_{test}|)) \end{aligned} \tag{7.16}$$

7.2.6 Experimental setup

The Markov chain method has been implemented in a framework entirely written in Java. The algorithm performance has been compared with Spectral feature alignment (SFA) by Pan et al. [PNS+10] and Joint sentiment-topic model with polarity-bearing topics (PBT) by He et al. [HLA11]. Pan et al. advanced a spectral feature alignment algorithm which aims to align words belonging to different domains into same clusters, by means of domain-independent terms. These clusters form a latent space which can be used to improve the sentiment classification accuracy on a target domain. He et al. extended the joint sentiment-topic model by adding prior words sentiment, thanks to the modification of the topic-word Dirichlet priors. Feature and document expansions are performed through adding polarity-bearing topics to align domains.

A common benchmark corpus is used in order to compare results, namely, a collection of Amazon reviews about Book (B), DVD (D), and Electronics (E). Each domain contains 1,000 positive and 1,000 negative reviews written in plain English. The text pre-processing phase described in Section 7.2.2 is applied to convert plain text into a bag-of-words. Then, before the learning phase and the classification phase introduced in Section 7.2.3 and Section 7.2.4 respectively, a feature selection is performed in accordance with one of the alternatives presented in Section 7.2.2. The goodness of results is measured by means of accuracy, averaged on 10 different source-target splits.

The approaches that will be compared below only differ in the applied feature selection method: document frequency, the opinion wordlist and the supervised feature selection technique with term ranking in Markov chain. Performance with every feature selection method is shown and compared with the state of the art.

7.2.7 Markov chain performance with different feature selection methods

In the first experiment, as shown in Figure 7.3, the only parameter to be set is the minimum document frequency df that a term must have in order to be selected. In other words, setting the minimum df to n means that terms occurring in less than n documents are ruled out from the analysis. This parameter has been varied from 25 to 100 with step length equal to 25. Each line in Figure 7.3 represents the trend for a specific source-target couple, namely, B $\rightarrow$ D, D $\rightarrow$ B, B $\rightarrow$ E, E $\rightarrow$ B, D $\rightarrow$ E, E $\rightarrow$ D. An extra line showing the average accuracy is also shown.

The accuracy decreases on average when the minimum document frequency increases. This outcome is probably due to the fact that document frequency is an unsupervised measure; therefore, considering smaller feature sets (i.e. higher df values) turns out not to be effective for

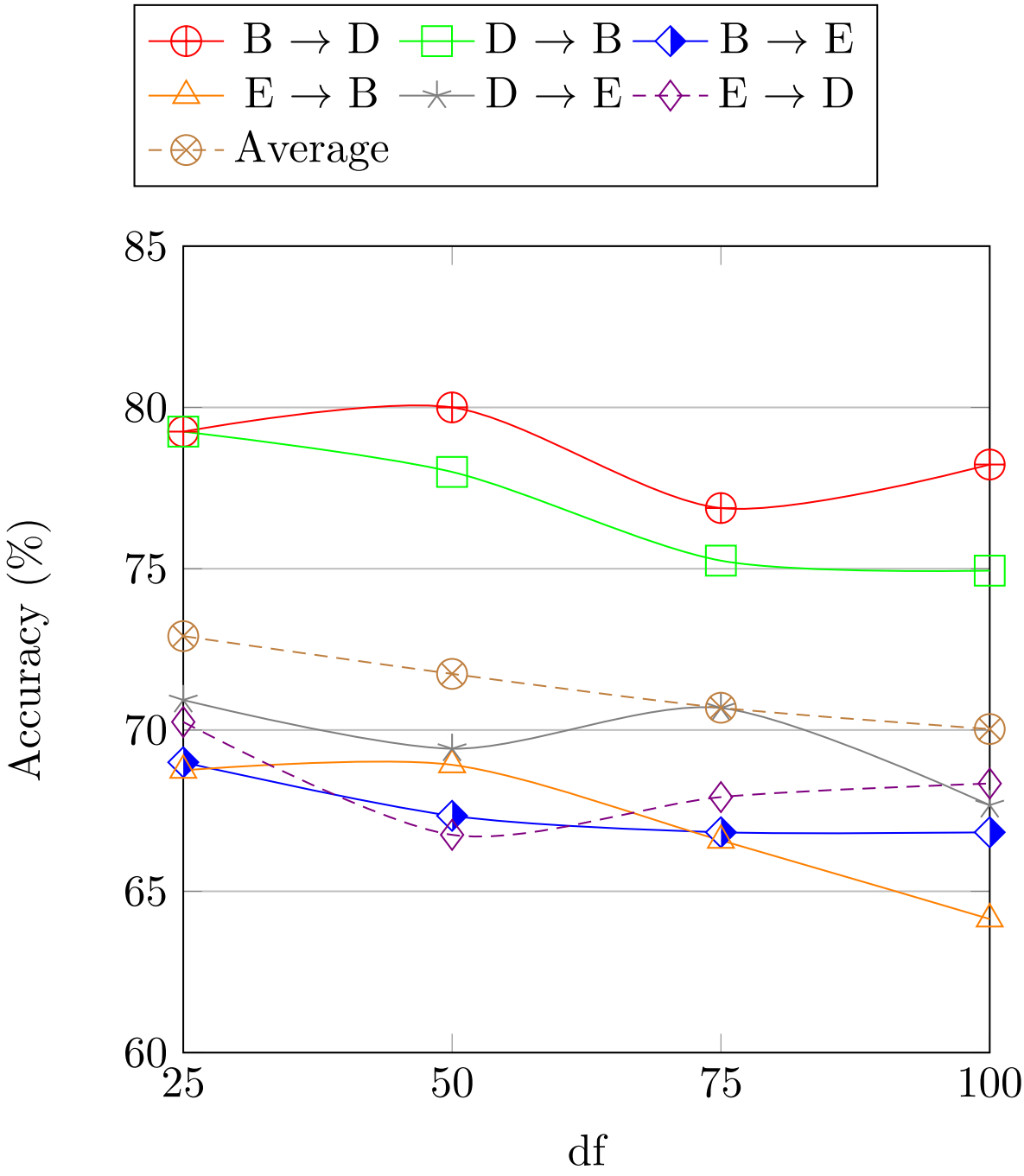


Figure 7.3: Cross-domain classification by varying the minimum df to be used for feature selection. *Source: [DMPP15].*

the classification of target documents. Unsupervised methods do not guarantee that the selected features help to discriminate among categories and, as such, a bigger dictionary could support the learning phase. Moreover, accuracy is lower anytime E is considered, supposedly because E is a completely different domain with respect to B and D, which instead have more words in common. The selection of the best feature set possible is really important in order to support knowledge transfer across domains.

An alternative to unsupervised methods consists in using as a dictionary the wordlist created by Bing Liu. No parameter needs to be tuned in this case, but Porter stemmer has to be applied to the wordlist so that the terms within documents match those in the wordlist. Comparing the algorithm performance when using opinion words as features with that achieved using terms having a minimum $df = 25$ (Figure 7.4), readers can note that the former outperforms the latter on average. This outcome suggests that transfer learning is fostered by features having stronger sentiment orientations. In such a case, the algorithm achieves better performance.

The wordlist by Bing Liu only contains opinion words, that are general terms with predefined polarities. Nevertheless, there may be other words, possibly domain-dependent, fostering the classification process. For this purpose, supervised techniques can be exploited to select the features to be used as a dictionary. It should be remarked that, to enhance the transfer learning capability of the method, information about sentiment should flow from source to target terms.

With reference to supervised feature selection techniques, two experiments are presented: in the former (Figure 7.5), features are selected by means of chi-squared (χ^2), varying the number of selected features from 250 to $1,000$ with step length 250; in the latter (Figure 7.4), the two supervised feature selection techniques mentioned in Section 7.2.2 are compared. Figure 7.5 reveals that there is no relevant variation on average by increasing the number of features. On one hand, this means that 250 words are enough to effectively represent the source-target couple, reminding that these are the best features according to the supervised method used. On the other hand, this proves that the Markov-based algorithm produces stable results by varying the number of features to be selected.

Figure 7.4 shows that the usage of a supervised feature selection technique is better on average than selecting just well-known opinion words. This outcome confirms the ability of the Markov chain method in performing transfer learning. Furthermore, the same configuration, where 250 features are selected by using χ^2 scoring function, also achieves better accuracy than using IG for feature selection (Figure 7.4). Summing up, the Markov chain method achieves its best performance on the Amazon corpus by selecting the best 250 features by means of χ^2 scoring function.

7.2.8 Comparison with the state of the art

Table 7.2 shows a comparison between the Markov chain method, referred as MC, Spectral feature alignment [PNS$^+$10] and Joint sentiment-topic model with polarity-bearing topics [HLA11], namely SFA and PBT. The accuracy is comparable, despite both SFA and PBT perform better on average. On the other hand, it deserves to be noted that MC only requires 250 features, whereas

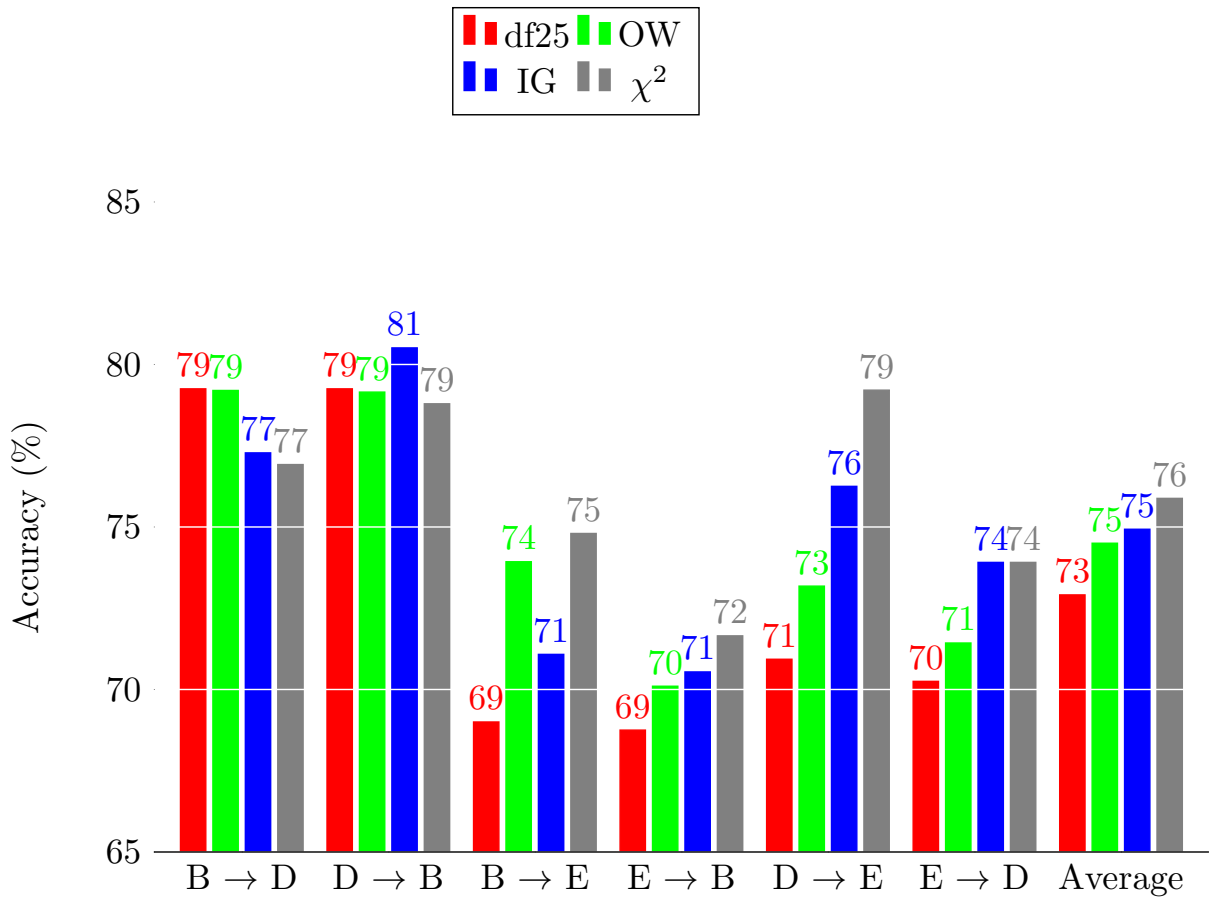


Figure 7.4: Cross-domain classification by comparing some feature selection methods, such as the unsupervised minimum document frequency df, the opinion wordlist by Bing Liu, and the supervised information gain IG and chi-square χ^2 scoring functions. A minimum $df = 25$ is set regarding the unsupervised method. Instead, the best 250 features are selected in accordance with the supervised scoring functions. *Source: [DMPP15].*

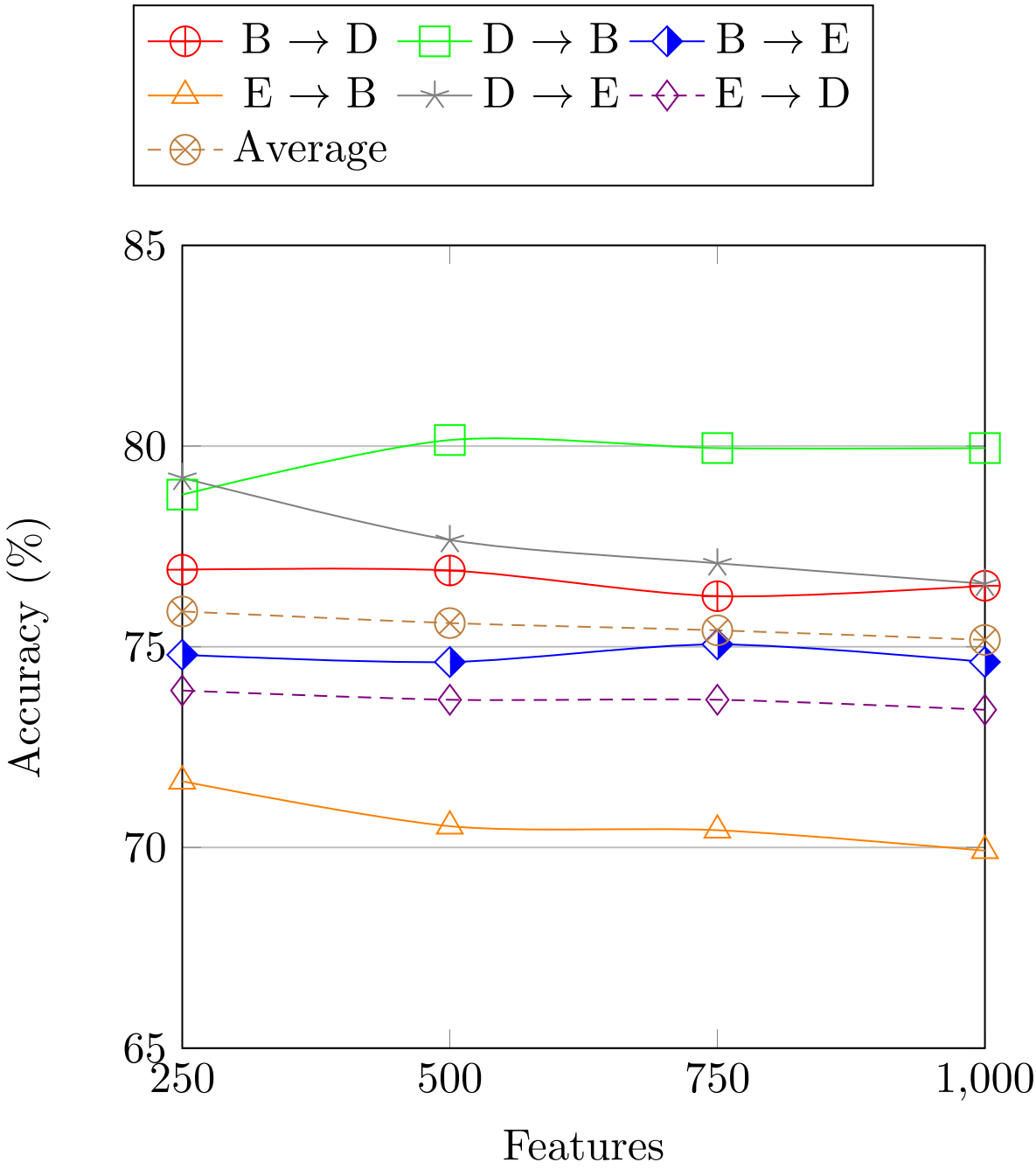


Figure 7.5: Cross-domain classification by varying the number of features to be selected in a supervised way. χ^2 scoring function is used in order to select features. *Source: [DMPP15].*

PBT needs $2,000$ features and SFA more than $470,000$, the size of datasets being equal. Since the computational complexity of all approaches quadratically grows with the number of features, the convergence of the proposed algorithm is supposed to be faster, although this hypothesis cannot be proved unless the other methods will be implemented.

Similar considerations can be done for in-domain tasks, as can be seen in Table 7.3. Readers should note that nothing needs to be changed in the Markov chain method to perform an in-domain sentiment classification, whereas other works use traditional classifiers completely bypassing the transfer learning phase.

Domain(s)	MC	SFA	PBT
B $\rightarrow$ D	76.92%	81.50%	81.00%
D $\rightarrow$ B	78.79%	78.00%	79.00%
B $\rightarrow$ E	74.80%	72.50%	78.00%
E $\rightarrow$ B	71.65%	75.00%	73.50%
D $\rightarrow$ E	79.21%	77.00%	79.00%
E $\rightarrow$ D	73.91%	77.50%	76.00%
Average	**75.88**%	**76.92**%	**77.75**%

Table 7.2: Cross-domain sentiment classification results, compared with other works.

Domain(s)	MC	SFA	PBT
B $\rightarrow$ B	76.77%	81.40%	79.96%
D $\rightarrow$ D	83.50%	82.55%	81.32%
E $\rightarrow$ E	80.90%	84.60%	83.61%
Average	**80.39**%	**82.85**%	**81.63**%

Table 7.3: In-domain sentiment classification results, compared with other works.

7.3 Variants of the Markov chain method

This section illustrates two variants of the method based on Markov chain theory described in Section 7.2.

The first modification considers documents as aggregates of sentences rather than as a whole block. The idea is that sentences are not guaranteed to bear the same sentiment orientation of the document within they are. Thus, classification can be performed sentence by sentence, then results can be combined in order to assign the final label to the document. The second modification aims at classifying the sentiment orientation of documents through polarity-bearing terms, which are those that contribute the most to the classification process. The idea is that a state transition is a function of the terms capability of predicting categories, whereas according to the method introduced in Section 7.2, they are only proportional to the semantic relationships between terms.

7.3.1 Document splitting into sentences

The first variant that will be illustrated considers the granularity of sentences rather than whole documents. The rationale is that documents can be composed of many sentences, possibly characterized by different sentiment orientations. It is not seldom to encounter documents expressing positive/negative opinions about a list of aspects, then ending with an overall opposite sentiment summarized in few words. Examples of the underlying behavior are *"My car's steering wheel always vibrates because of [...] the seat is not so comfortable [...] but in general I like my car."* or even *"I read that book. Characters are well described, their psychological profile is meticulously portrayed. However, the plot is definitely boring and I always fall asleep while reading!"*

During the text pre-processing phase, documents are split into sentences using the set of characters $Tok = \{.,;,!,?\}$ as tokenizers. During this process, some exceptions may occur that can invalidate the splitting. Only the most straightforward cases have been handled, like for example *Ph.D.*, *Dr.*, websites and emails. Anyway readers should be aware that, depending on the training set, many nontrivial cases might negatively affect the splitting.

According to this method variant, co-occurrences between terms are no longer to be evaluated within the same document, but within the same sentence. More formally, considering each document d_z as a set of sentences $d_z = \{p_1^z, p_2^z, \ldots, p_h^z\}$, Equation (7.4) can be rewritten as follows:

$$a_{ij} = a_{ji} = \sum_{d \in \mathscr{D}} \sum_{p \in d} w_{t_i}^p \cdot w_{t_j}^p \tag{7.17}$$

Consequently, Equation (7.6) becomes

$$a_{ij} = a_{ji} = \begin{cases} 0, & i = j \\ \sum\limits_{d \in \mathscr{D}_{train} \cup \mathscr{D}_{test}} \sum\limits_{p \in d} w_{t_i}^p \cdot w_{t_j}^p, & i \neq j \end{cases} \tag{7.18}$$

Similarly, the transition from a term to a class in Equation (7.5) can be rewritten as

$$b_{ij} = \sum_{d \in \mathscr{D}} \sum_{p^j \in d} w_{t_i}^{p^j} \tag{7.19}$$

where $p^j = \{p_t \in d : c_{p_t} = c_j\}$. Likewise, Equation (7.7) comes to be

$$b_{ij} = \sum_{d \in \mathscr{D}_{train}} \sum_{p^j \in d} w_{t_i}^{p^j} \tag{7.20}$$

In the classification phase, Equation (7.11) and Equation (7.12) have been modified in order to label the single sentences inside a document rather than the document itself:

$$p_t^* = (w_{t_1}^{p_t^*}, \ldots, w_{t_k}^{p_t^*}, c_1^*, \ldots, c_M^*) = p_t \times MCTM \tag{7.21}$$

$$c_{p_t} = \arg\max_{i \in C^*} c_i^* \tag{7.22}$$

where p_t is a column vector having size $(k+M)$.

The output labels for each sentence are finally combined by voting in order to obtain the final category for the document to be classified:

$$c_d = \arg\max_{i \in \mathscr{C}} |c_i^d| \tag{7.23}$$

where $|c_i^d| = |\{p_t \in d : c_{p_t} = c_i\}|$. The computational complexity is now a function of the cardinality of sentences in the corpus, rather than of the documents cardinality as in the basic method.

7.3.2 Polarity-driven state transitions

A second alternative, orthogonal to the previous one, has been developed to establish how much a term should be linked with the others and with classes. For this purpose, it should be taken into account that the probability the current state has at time t will be redistributed to other states at time $t+1$. Such redistribution takes place based on co-occurrences so far, as stated by Equation (7.6) and Equation (7.7).

Although it apparently is reasonable, this mechanism is not effective enough because the capability of states to discriminate among categories is completely overlooked during state transitions. Issues may occur for such a reason. For instance, if the current state is not well polarized, not only it will not be able to distinguish among classes, but it will likely be connected to terms having conflicting sentiment. Moreover, if a term semantically related to the current state is not well polarized, it should not be selected as a future state because it will not be useful for classification.

In order to solve these issues, the focus should be on what occurs during state transitions. The intuition is that the more the current state is capable to discriminate among categories (i.e. it is a polarity-bearing term) the more its probability will be distributed to classes. The remaining part will be allotted among the other terms, proportionally not only to the semantic relationship between the current state and the future state, but also to the capability of the latter to distinguish among classes. No further expedient is required to fulfil this double goal, other than Markov chain. In fact, the capability f_i of a term t_i to discriminate among categories can be defined as:

$$f_i = \frac{|b_{i+} - b_{i-}|}{b_{i+} + b_{i-}} \tag{7.24}$$

where $0 \leq f_i \leq 1$, whereas b_{ij} has been defined by Equation (7.7). In other words, f_i is the portion of probability that t_i will redistribute to classes in a proportional way to the values computed by Equation (7.7). This means that the polarity-bearing terms are those that contribute the most

to the classification process. The remaining $(1 - f_i)$ will be split among terms according to the following relation:

$$a_{ij} = \begin{cases} 0, & i = j \\ f_j \cdot \sum\limits_{d \in \mathscr{D}_{train} \cup \mathscr{D}_{test}} w_{t_i}^d \cdot w_{t_j}^d, & i \neq j \end{cases} \tag{7.25}$$

Note that the transition probability depends not only on the semantic relationship among terms, but also on the capability of the destination term to detect categories.

It deserves to be reminded that, according to Equation (7.10), when classifying a document d_t the initial state of Markov chain is represented by $w_{t_1}^{d_t}, \ldots, w_{t_k}^{d_t}$. Each weight w_{t_i} has to be multiplied by f_i as well, since only the polarity-bearing terms should drive the sentiment classification, spreading their probability during Markov chain state transitions. The overall computational complexity is aligned with the basic approach shown in Section 7.2.5.

7.3.3 Analysis and results

The performance of all variants is assessed and compared with the state of the art. The text pre-processing phase described in Section 7.2.2 is applied to convert plain text into the bag-of-words representation, possibly splitting documents into sentences when dealing with the variant introduced in Section 7.3.1. Before the learning and classification phases, feature selection is performed by means of χ^2, which turned out to be the best performing technique for this purpose, as shown in Figure 7.4.

From now on, MC_S will be used for the Markov chain variant characterized by splitting documents into sentences, MC_P for the polarity-driven transitions one, MC_{SP} for their combination, whereas MC_B will stand for the basic approach.

In order to compare performance with MC_B, the experiment that led to the best accuracy has been replicated. Therefore, the training set is composed of $1,600$ documents and the test set of 400 documents. The best 250 features are selected during text pre-processing by means of χ^2 scoring function. The goodness of results is measured by means of accuracy, averaged on 10 random source-target splits, and evaluated for each particular source-target combination, namely B $\rightarrow$ D, D $\rightarrow$ B, B $\rightarrow$ E, E $\rightarrow$ B, D $\rightarrow$ E, E $\rightarrow$ D, even including in-domain configurations, such as B $\rightarrow$ B, D $\rightarrow$ D, E $\rightarrow$ E.

As can be noted in Figure 7.6, both the variants relying on sentence splitting (i.e. MC_S and MC_{SP}) do not perform well. The reason for this outcome is to be found in the learning phase, where the polarity of each sentence should be taken into account, as stated by Equation (7.19). However, only the sentiment orientation of the whole document is available in the Amazon corpus, whereas the polarity at the sentence level is not available. Each sentence in a training document was supposed to have the same sentiment of the document itself. This is a strong assumption, definitely false in general. Such an assumption may lead to wrongly consider the sentiment at the term level and, as a consequence, to bad performance.

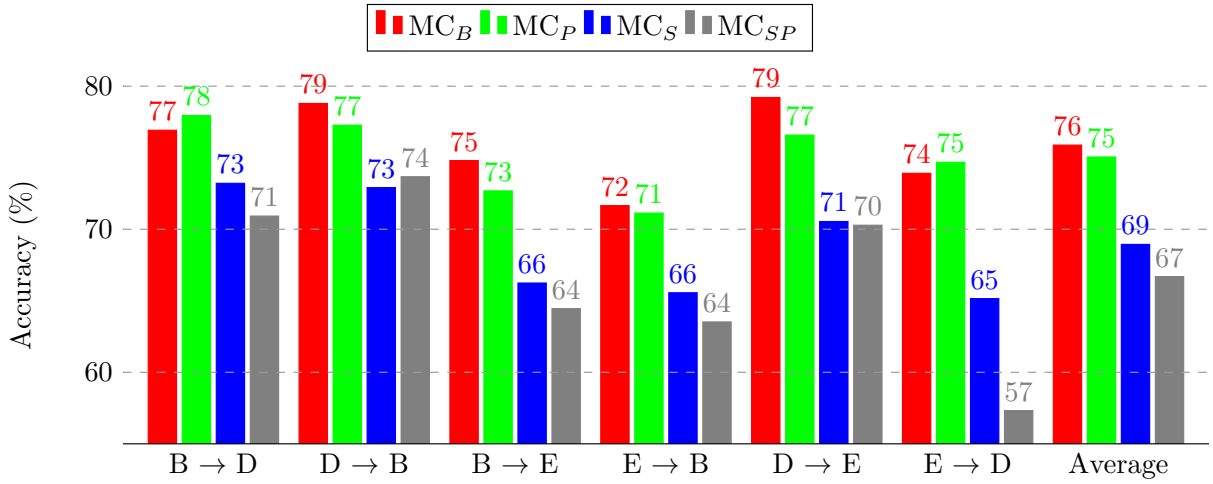


Figure 7.6: Cross-domain classification by comparing all the proposed Markov chain variants. The best 250 features are selected in accordance with the score output by χ^2. *Source: [DMPP].*

A qualitative comparison between MC$_P$ and MC$_B$ can be seen in Figure 7.7. The reported examples show that polarity-driven state transitions could be helpful for the classification process when there are some polarity-bearing terms within the document to be classified. In such cases, the MC$_P$ classifier is much more confident with its prediction than MC$_B$. This could also bring (as in example B) to correctly predict test instances failed by the basic classifier.

Although MC$_P$ is able to take advantage of polarity-bearing terms, its accuracy does not outperform that of MC$_B$. This outcome could be explained considering that there is no constraint that forces terms to redistribute their probability to others having the same sentiment. A document that does not contain most terms bearing its own polarity is likely to be misclassified because, although terms are semantically related, they separately contribute to classification. This happens for example when a positive/negative document expresses opposite opinions about some aspects before summarizing in few words its overall polarity. The classification phase initially assumes to be in different states at the same time, corresponding to the terms within the document. Then, each state evolves step by step independently of the others; state evolution is only determined by the semantic relationships between terms, learned when training the model. Context is neglected during classification.

Table 7.4 shows a comparison between the Markov chain based methods and other works, namely SFA and PBT. Whereas MC$_S$ and MC$_{SP}$ are far away from the state of the art, MC$_B$ and MC$_P$ achieve comparable results, despite both SFA and PBT perform better on average. On the other side, it should be reminded that the proposed algorithms require much fewer features than the others, i.e. 250 with respect to the $2,000$ needed by PBT and the more than $470,000$ needed by SFA, the size of datasets being equal. Therefore, since the computational complexity of all techniques quadratically grows with the number of features, the convergence of both MC$_B$ and MC$_P$ is supposed to be dramatically faster than that of SFA and PBT. This makes the proposed

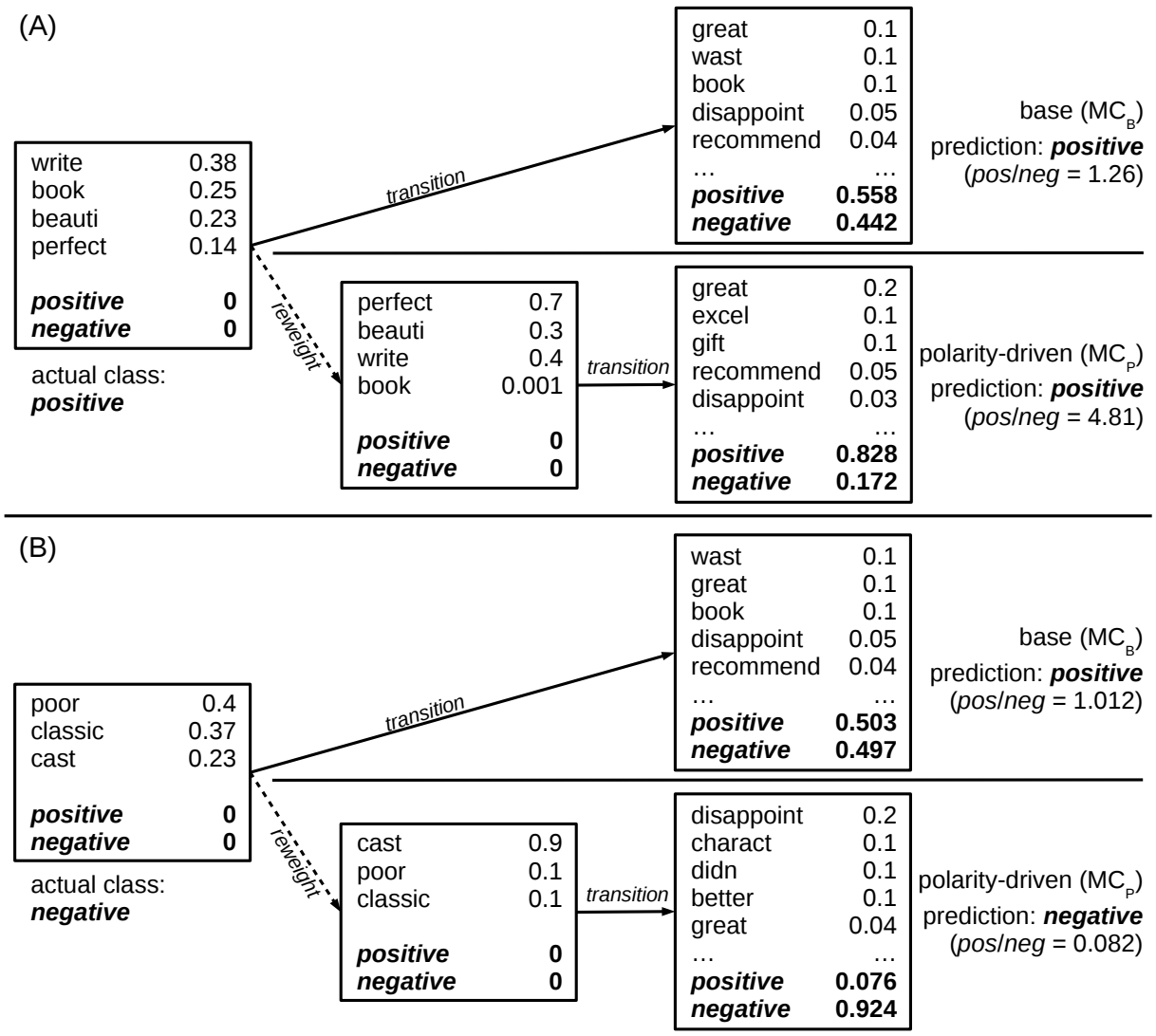


Figure 7.7: Two examples of documents (A and B), represented as selected features with associated weights, classified by using either MC_B or MC_P. Each of the rightmost boxes only shows the 5 terms with the highest weights after the transition. *Source: [DMPP].*

algorithms suitable for big data scenarios.

Lastly, similar considerations can be done for in-domain tasks. Nothing needs to be changed in order to perform in-domain sentiment classification, whereas other works use standard classifiers completely bypassing the transfer learning phase.

	Other methods		Markov chain method variants			
Domain(s)	SFA	PBT	MC_S	MC_{SP}	MC_B	MC_P
Cross-domain experiments (*source* → *target*)						
B → D	81.50%	81.00%	73.21%	70.92%	76.92%	77.95%
D → B	78.00%	79.00%	72.91%	73.67%	78.79%	77.27%
B → E	72.50%	78.00%	66.24%	64.45%	74.80%	72.68%
E → B	75.00%	73.50%	65.56%	63.52%	71.65%	71.13%
D → E	77.00%	79.00%	70.54%	70.28%	79.21%	76.58%
E → D	77.50%	76.00%	65.15%	57.32%	73.91%	74.68%
Average	**76.92%**	**77.75%**	**68.94%**	**66.69%**	**75.88%**	**75.05%**
In-domain experiments						
B → B	81.40%	79.96%	73.72%	72.45%	76.77%	77.78%
D → D	82.55%	81.32%	78.34%	79.60%	83.50%	82.49%
E → E	84.60%	83.61%	77.78%	78.54%	80.90%	79.55%
Average	**82.85%**	**81.63%**	**76.61%**	**76.86%**	**80.39%**	**79.94%**

Table 7.4: In-domain and cross-domain sentiment classification performance of all the Markov chain based methods, compared with other works.

7.4 Final remarks

In this chapter, we have introduced some techniques based on Markov theory to perform both sentiment classification and transfer learning in cross-domain tasks. The basic algorithm aims at building a Markov chain transition matrix, whose states represent either terms or classes, and whose transitions depend on term co-occurrences in documents. A first variant is proposed that aims at considering the granularity of sentences rather than perceiving documents as a whole. A second modification affects the classification process, so that it can be driven by polarity-bearing terms.

Experiments on a common benchmark corpus show that the proposed algorithm achieves performance comparable with the state of the art with respect to both in-domain and cross-domain sentiment classification. Also, the introduced techniques require less parameters to be tuned. Finally, in spite of having a comparable computational complexity, which quadratically grows with the number of features, much fewer terms are required to achieve good accuracy. The last two characteristics mentioned make our Markov-based techniques suitable for the analysis of big datasets.

8

Deep methods to enhance text understanding in big dataset analysis

Traditional transfer learning techniques are not appropriate for the analysis of large datasets. In spite of their good performance with small size data, they do not scale properly with the cardinality of datasets, principally due to computational complexity and high-dimensional text representation.

In this chapter, an analysis of distributed text representation methods is performed, explaining how they improve text understanding and discussing their impact on transfer learning. Another deep learning progress is then examined, namely memory-based deep neural networks, showing that some deep architectures are automatically able to store and preserve relevant information in memory through time. This capability makes memory-based deep neural networks very appropriate for both in-domain and cross-domain sentiment classification. Finally, we show how such recent deep learning advances can be combined to have a breakthrough in cross-domain sentiment classification.

8.1 Deep learning

Before focusing on distributed text representations and memory-based deep neural networks, we make a brief introduction of deep learning, explaining its general characteristics, and discussing its impact on sentiment classification.

8.1.1 Overview

Conventional machine learning techniques have been applied to a large variety of tasks, including the identification of object in images, speech-to-text translation, information retrieval, matching problems, and so on. However, traditional approaches have a limited ability to process raw data. To deal with this problem, the approach used for a long time was to design a feature extractor able to map raw data into a more suitable representation. This is effective, but has some drawbacks: first, a feature extractor needs to be carefully engineered; and second, domain expertise is needed in order to design it correctly.

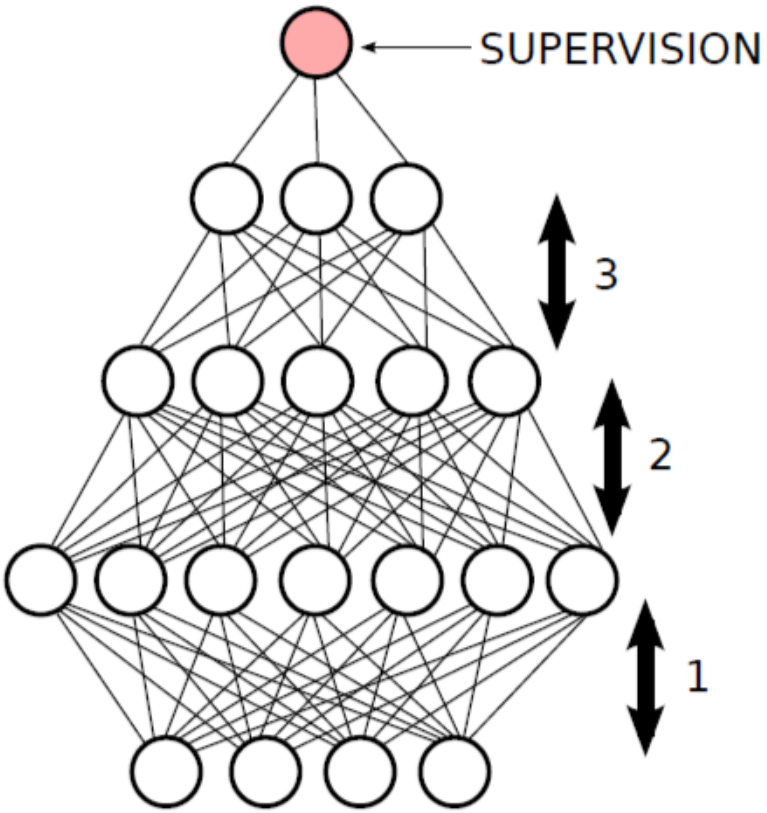


Figure 8.1: A general deep learning architecture.

An alternative approach is representation learning, whose goal is the automatic discovery of a suitable representation from raw data. Deep learning techniques are representation learning methods with multiple levels of representation, as can be seen in Figure 8.1. Each layer is obtained by composing different modules, which generally involve the application of simple functions. Very often these functions are non-linear, like for example the hyperbolic tangent or the sigmoid function. Each module transforms the representation at one level into a representation at a higher, more abstract level. The layers of deep neural networks are typically trained without supervision, which is generally added just in the upper layer of the network. This brings to robust data representations, independent of the task.

Basically, deep architectures are able to learn complex functions by composing simpler functions, extracting intricate structures and patterns in high-dimensional data. There is no need for feature engineering, because a suitable data representation is automatically learned. Moreover, deep architectures are scalable in terms of data, because larger amount of data brings to better representations, and they are less affected by the curse of dimensionality.

8.1.2 The impact on sentiment classification

Thanks to such advantages, deep learning is recently having a great impact on several research areas, such as image recognition and classification, speech processing and recognition, visual object detection, video, audio, natural language processing, machine translation, drug discovery and genomics [LBH15]. Here we only focus on sentiment classification, but careful readers can find further details in [LBH15, B$^+$09, GBCB16, Sch15].

Deep learning brought to a dramatic improvement in sentiment classification. Socher et al. [SPW$^+$13] introduced Recursive Neural Tensor Networks to foster single sentence sentiment classification. Apart from the high accuracy achieved in classification, these networks are able to capture sentiment negations in sentences due to their recursive structure. Dos Santos et al. [dSG14] proposed a Deep Convolutional Neural Network that jointly uses character-level, word-level and sentence-level representations to perform sentiment analysis of short texts. Kumar et al. [KIO$^+$16] presented Dynamic Memory Network (DMN), a neural network architecture that processes input sequences and questions, forms episodic memories, and generates relevant answers. The ability of DMNs to naturally capture position and temporality allows this architecture achieving the state-of-the-art performance in single sentence sentiment classification over the Stanford Sentiment Treebank proposed in [SPW$^+$13]. Tang et al. [TQL15] introduced Gated Recurrent Neural Networks to learn vector-based document representation, showing that the underlying model outperforms the standard Recurrent Neural Networks in document modeling for sentiment classification. Zhang and LeCun [ZL15a] applied temporal convolutional networks to large-scale data sets, showing that they can perform well without the knowledge of words or any other syntactic or semantic structures. Wang et al. [WJL16] combined Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) for sentiment analysis of short texts, taking advantage of the coarse-grained local features generated by CNN and long-distance dependencies learned via RNN. Chen et al. [CXH$^+$16] proposed a three-steps approach to learn a sentiment classifier for product reviews. First, they learned a distributed representation of each review by a one-dimensional CNN. Then, they employed a RNN with gated recurrent units to learn distributed representations of users and products. Finally, they learned a sentiment classifier from user, product and review representations.

Despite the recent success of deep learning in in-domain sentiment classification tasks, few attempts have been made in cross-domain problems. Glorot et al. [GBB11] used Stacked Denoising Autoencoder introduced in [VLL$^+$10] to extract domain-independent features in an unsupervised fashion, which can help transferring the knowledge extracted from a source domain to a target domain. However, they relied only on the most frequent $5,000$ terms of the vocabulary for computational reasons. Although this constraint is often acceptable with small or medium data sets, it could be a strong limitation in big data scenarios, where very large data sets are required to be analyzed.

8.2 Distributed text representation for transfer learning and cross-domain

As already shown in Section 3.2.3, many transfer learning approaches have been proposed to bridge the semantic gap between heterogeneous domains. Unfortunately, non-deep transfer learning solutions have several drawbacks. First of all, they require complex human engineering, for instance to identify polarity-bearing terms, to identify common terms that act as a bridge between source and target, and to find the best way for knowledge transfer. A second limitation is that they are generally based on bag-of-words representations. Bag-of-words intrinsically loses the ordering of the words and, as a consequence, fails to capture semantics. Moreover, bag-of-words is affected by the curse of dimensionality, because every term becomes a feature of the model. Some transfer learning techniques rely on external knowledge bases, but often they are not enough to align heterogeneous domains. Finally, one of the biggest problems is that non-deep transfer learning methods require many parameters to be tuned, because the best values for a given dataset often are not equally good for a different dataset. This is not only an issue related to time complexity, but actually when data are unlabeled there is no way to perform tuning and find optimal values.

On the other hand, there are several reasons why deep learning might help transfer learning. Since deep learning is a special case of representation learning, a suitable text representation can automatically be discovered from data. Deep methods are generally based on distributed word representations, which are able to preserve word ordering and to capture word semantics without supervision. In distributed word representations, terms are mapped into a low-dimensional space. This improves scalability and paves the way for the use of large-scale data, essential in big data scenarios.

Supported by these considerations, this section analyzes the impact of distributed text representations on the alignment of heterogeneous domains. In particular, we investigate if, and to what extent, deep learning algorithms can automatically bridge the inter-domain semantic gap typical of cross-domain sentiment classification, strengthened by their ability to learn syntactic as well as possibly hidden relationships in text. This investigation is also sustained by several works [LM14, SPW+13, TQL15, ZL15a], which pointed out the ability of deep approaches to learn semantic-bearing word representations, typically without supervision and independently of domains.

To assess the potentiality of deep learning in cross-domain sentiment classification, two approaches are compared: the Markov chain method introduced in Chapter 7, and paragraph vectors for distributed representations of sentences and documents [LM14]. As discussed in Chapter 7, the Markov chain approach is tailored to cross-domain sentiment classification, where it has achieved performance comparable with the state-of-the-art on some benchmark text sets. Paragraph vector is a well-known unsupervised deep learning method that is able to map words into a vector space wherein semantics arises, that is, similar words are closer than unrelated ones. Although paragraph vector has not been designed for cross-domain sentiment classification and does not provide a transfer learning phase, its ability to learn semantic-bearing

word representations might help bridging the inter-domain semantic gap.

To assess such an idea, in-domain experiments are firstly performed, so as to have a baseline for cross-domain analysis. Then, the ability of both approaches to transfer knowledge across heterogeneous domains is evaluated and discussed.

8.2.1 Methods

Here the main features of the two techniques are described, pointing out the characteristics that may foster cross-domain sentiment classification.

Paragraph vector

Paragraph vector is an unsupervised deep learning method that aims at solving the weaknesses of bag-of-words models. Alike bag-of-words, paragraph vector learns fixed-length feature representation from variable-length pieces of texts, such as sentences, paragraphs, and documents. However, bag-of-words features lose the ordering of the words and, as a consequence, they fail to capture semantics. For example, *good*, *robust* and *town* are equally distant in the feature space, despite the fact that *good* should be closer to *robust* than *town* from a semantic point of view. The same issues occur to the bag-of-n-grams model. In spite of considering the word ordering in short contexts, it suffers from data sparsity and high dimensionality.

On the other hand, paragraph vector intrinsically handles the word order representing each document by a dense vector, which is trained to predict words in the document itself. More precisely, the paragraph vector is concatenated with some word vectors from the same document to predict the following word in the given context. The paragraph token can be thought of as another word that acts as a memory, remembering what is missing from the current context. For this reason, this model, represented in Figure 8.2, is called the Distributed Memory Model of Paragraph Vector (PV-DM). Another way to learn the paragraph vector is to ignore the context words in input, forcing the model to predict words randomly sampled from the paragraph. At each iteration of the stochastic gradient descent, a text window is sampled, then a random word is sampled from the text window and a classification task is formed given the paragraph vector. This version of the paragraph vector, shown in Figure 8.3, is called the Distributed Bag of Words version of Paragraph Vector (PV-DBOW). Both word vectors and paragraph vectors are trained by means of the stochastic gradient descent and backpropagation [RHW86].

Sentiment classification requires sequential data to be handled, because the document semantics is typically affected by the word order. Paragraph vector has been shown to learn a vector representation for such sequential data, becoming a candidate technique for sentiment classification. Readers should remember that paragraph vector learns fixed-length feature representation from variable-length pieces of texts, dealing with any kind of plain text, from sentences to paragraphs, to whole documents. Though, this aspect is just as relevant as knowing how many of these features are actually required to learn accurate models. Feature vectors have dimensions in the order of hundreds, much less than bag-of-words representations, where every

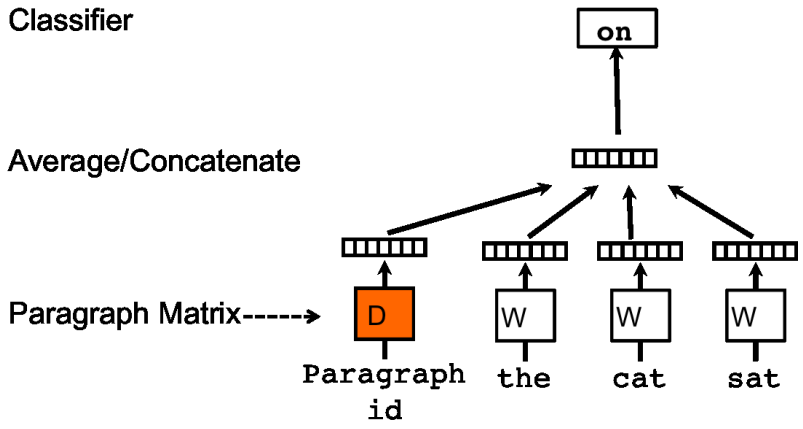


Figure 8.2: A framework to learn the Distributed Memory Model of Paragraph Vector (PV-DM). With respect to word vectors, an additional paragraph token is mapped to a vector via matrix D. In such a model, the concatenation or average of this vector with a context of three words is used to predict the fourth word. The paragraph vector represents the missing information from the current context and can act as a memory of the topic of the paragraph. *Source: [LM14].*

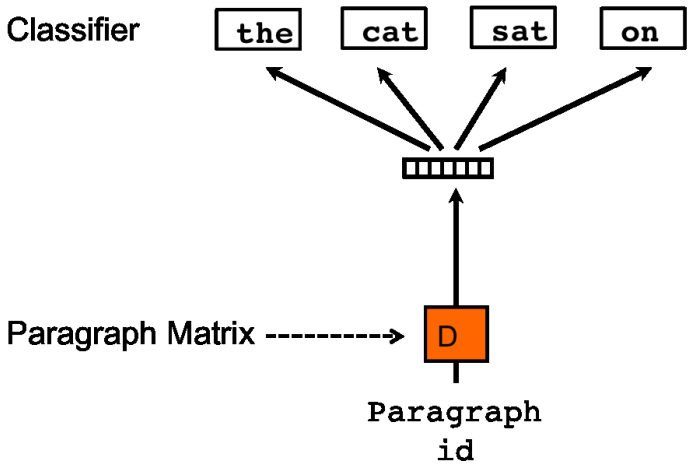


Figure 8.3: Distributed Bag of Words version of Paragraph Vector (PV-DBOW), where the paragraph vector is trained to predict the words in a small window. *Source: [LM14].*

term becomes a feature of the model. The consequence is that either bag-of-words models cannot be used to represent very large data sets due to the huge number of features or a feature selection is needed to reduce dimensionality. Feature selection entails information loss, beyond requiring a tuning phase to choose the right number of features. The fact that paragraph vector is not affected by the curse of dimensionality suggests that the underlying method is not only scalable with large data sets, but also it entirely preserves information by increasing the data set size.

In [LM14], paragraph vector has been shown to achieve brilliant in-domain sentiment classification results, but no cross-domain experiment has been conducted. Nevertheless, some characteristics of paragraph vector make it suitable for cross-domain sentiment classification, where the language is usually heterogeneous across domains. Paragraph vector is very powerful in modeling syntactic as well as hidden relationships in plain text without any kind of supervision. Words are mapped to positions in a vector space wherein the distance between vectors is closely related to their semantic similarity. The capability of extracting both word semantics and relationships in an unsupervised fashion makes it interesting to test whether paragraph vector is able to automatically handle transfer learning. For this purpose, a comparison with the Markov chain method will be shown later on in Section 8.2.2.

As described in [LM14], in order to use the available labeled data, each subphrase is treated as an independent sentence and the representations for all the subphrases in the training set are learned. After learning vector representations for the training sentences and their subphrases, they are fed to a logistic regression to learn a predictor of the sentiment orientation. At the test time, the vector representation for each word is frozen, and the representations for sentences are learned using the stochastic gradient descent. Once the vector representations for the test sentences are learned, they are fed through the logistic regression to predict the final label.

Markov chain

Alike paragraph vector, the Markov chain can handle sentences, paragraphs and documents, but it is much more affected by the curse of dimensionality, because it is based on a dense bag-of-words model. Feature selection is often advisable to mitigate this issue, or even necessary with very large data sets, typically containing million or billion words. Only the k most significant terms according to a given scoring function are kept.

Readers should recall the basic idea of the Markov chain approach, that consists in modeling term co-occurrences: the more terms co-occur in documents the more their connection are stronger. The same strategy could be followed to model the polarity of a given term: the more terms are contained in positive/negative documents the more they tend to be positive/negative. Following this idea, terms and classes are represented as states of a Markov chain, whereas term-term and term-class relationships are modelled as transitions between these states.

Thanks to this representation, the Markov chain introduced in Section 7.2 is able to perform both sentiment classification and transfer learning. On one hand, it can be used as a classifier because classes are reachable from terms at each Markov chain state transition, since each edge models a term-class relationship. On the other hand, it assumes that there exists a subset of

common terms between the source and target domains that act as a bridge between domain specific terms, allowing and supporting transfer learning. Dealing with such an assumption, at each state transition in the Markov chain, sentiment information can flow from source-specific to target-specific terms passing through a layer of shared terms (Figure 7.2). Actually, the classification process usually works in the opposite direction, i.e. from target-specific to source-specific terms, and goes on while the class states are eventually reached.

For instance, say that a review from the target domain only contains target-specific terms. None of these terms is connected to the classes, but they are connected to some terms within the shared terms, which in turn are connected to some source-specific terms. Both the shared and source-specific terms are connected to the classes. Therefore, starting from some target-specific terms, first the Markov chain performs transfer learning and then sentiment classification. It should be remarked that the transfer learning mechanism is not an additional step to be added in cross-domain tasks; on the contrary, it is intrinsic to the Markov chain algorithm.

Careful readers can find further details on the Markov chain method in Chapter 7, and on paragraph vectors in [LM14].

8.2.2 Experimental setup

Some experiments have been performed to show whether outstanding unsupervised techniques as the paragraph vector are suitable for cross-domain sentiment classification, despite no explicit mechanism to manage transfer learning. Also, the underlying investigation gives users insights into the awkward choice of the most suitable algorithm for a given problem, with reference to the amount of data available for training.

While the Markov chain (MC) method has been implemented in a custom Java-based framework, for paragraph Vector (PV) *gensim* [RS10] has been used, a Python-based open sourced and freely available framework.[1] In particular, tests have been performed by using 0.12.4 gensim release. Apart from the two main approaches compared, Naïve Bayes (NB) has also been employed as baseline for the experiments. The implementation is from the 3.9.1 software release of the Weka [FHH+05] workbench.

In order for the results to be comparable, a common benchmark corpus has been chosen, namely, a collection of Amazon reviews[2] about Books (B), Movies (M), Electronics (E) and Clothing-Shoes-Jewelry (J). Each domain contains plain English reviews along with their labels, namely a score from 1 (very negative) to 5 (very positive). Focusing on binary sentiment classification, where only positive and negative classes are considered, the five original categories have to be transformed. The reviews with scores equal to 1 and 2 are mapped to the negative category, those whose scores is 4 and 5 to the positive one, discarding those whose score is 3 because of their neutral sentiment orientation.

For the sake of assessing the algorithms effectiveness by varying the amount of labeled data available to train the models, source-target partitions with three different orders of magnitude

[1] http://nlp.fi.muni.cz/projekty/gensim/

[2] `http://jmcauley.ucsd.edu/data/amazon/`

have been tested, always preserving the source-target ratio, i.e. $80\% - 20\%$, and the balancing between positive and negative instances. The smallest data set counts $1,600$ instances as the training set and 400 as the test set; the medium $16,000$ and $4,000$ respectively; and the largest $80,000$ and $20,000$ respectively. For each examined source-target combination, accuracy has been chosen as the performance measure, namely, the percentage of correctly classified instances. This is a typical choice in a balanced binary classification problem, where the same number of instances is considered for both categories. Moreover, results have been averaged on 10 different training-test partitions to reduce the variance, that is, the sensitivity to small fluctuations in the training set.

In the following experiments, the Distributed Bag of Words version of Paragraph Vector (PV-DBOW) has been used, choosing 100-dimensional feature vectors, considering 10 words in the window size, ignoring words occurring in just one document and applying negative sampling with 5 negative samples. Moreover, the initial learning rate has been set to 0.025, letting it linearly drop to 0.001 in 30 epochs. For further details on the parameters, careful readers may refer to [LM14, MSC+13]. To accomplish sentiment classification, the positive or negative orientation of reviews has been predicted by means of a logistic classifier, whose regression coefficients have been estimated employing the Newton-Raphson method.

Concerning the Markov chain method, experiments have been conducted on the technique described in Section 7.2. Firstly, the relative frequency of terms in documents has been chosen as the term weighting. Then, a subset of relevant features is selected by means of χ^2 scoring function to mitigate the curse of dimensionality, because the Markov chain method is based on a dense bag-of-words model, as explained in Section 8.2.1. After a tiny tuning, 750, $10,000$ and $25,000$ terms have been chosen for the small, medium and large data sets respectively. The Markov chain has been built including the selected terms only. As already explained in Section 8.2.1, the more terms co-occur in reviews the more their connection are stronger. Likewise, the more terms are in positive (negative) reviews the more they tend to be positive (negative). For further details, readers may refer to Chapter 7.

Finally, Naïve Bayes has been run with default parameters after the same feature selection performed for Markov chain, namely, 750, $10,000$ and $25,000$ terms for the small, medium and large data sets respectively by means of χ^2 scoring function.

Three experiments will be shown below. The first focuses on in-domain sentiment classification, namely, where the algorithms are tested on a set of reviews from the same domain used for model training. In this way, an evaluation can be performed of how much performance varies with respect to the amount of training data. The second experiment focuses on cross-domain sentiment classification, where transfer learning is usually needed to walk around the heterogeneity of language across domains. This analysis assesses the capability of paragraph vector to automatically bridge the inter-domain semantic gap, without providing any explicit transfer learning mechanism. The last experiment analyzes whether or not a simple multi-source approach positively affects the performance of paragraph vector in cross-domain sentiment classification.

8.2.3 In-domain analysis

The first experiment assesses the in-domain performance of the algorithms. Table 8.1 shows an analysis of the 4 domains considered, namely Books (B), Movies (M), Electronics (E) and Clothing-Shoes-Jewelry (J).

	1.6k-0.4k			16k-4k			80k-20k		
Domain(s)	PV	MC	NB	PV	MC	NB	PV	MC	NB
In-domain experiments									
B → B	67.25%	79.25%	78.25%	75.40%	81.90%	63.30%	84.74%	83.84%	66.36%
M → M	79.75%	91.23%	74.50%	74.87%	82.43%	70.10%	84.11%	80.23%	71.30%
E → E	79.25%	92.00%	79.50%	80.15%	80.72%	69.53%	85.61%	84.41%	76.41%
J → J	75.50%	71.97%	67.50%	80.08%	83.76%	72.23%	85.25%	86.98%	73.81%
Average	**75.44%**	**83.61%**	**74.94%**	**77.63%**	**82.20%**	**68.79%**	**84.93%**	**83.87%**	**71.97%**

Table 8.1: Comparison between PV and MC in sentiment classification, using NB as a baseline. Nk-Mk means that the experiment has been performed by using N*1000 instances as the training set and M*1000 instances as the test set. X → Y means that the model has been learned on reviews from domain X and then applied to different reviews from domain Y.

The first outcome that catches the eye is that paragraph vector requires much more training data than Markov chain in order to perform well, as it is even clearer by observing Figure 8.4. Although it achieved brilliant results as stated by Le and Mikolov [LM14], it underperforms Markov chain and performs slightly better than Naïve Bayes on average when small data sets are involved. This outcome is not completely surprising because deep learning techniques typically require very large training sets to learn models that are able to generalize over new test instances. On the other hand, paragraph vector scales very well in terms of accuracy when the model is learned from very large labeled text sets. Accuracy boosts from 75.44% in the small text set to 77.63% with medium-scale data, reaching 84.93% on average in the largest. Careful readers might have noted that, when enough training data are available, the accuracy of paragraph vector has a low deviation from the mean in each of the four domains. This proves that paragraph vector is a robust approach, which is effective independently of the particular domain analyzed.

Also, it is noteworthy to point out that the accuracy achieved by Markov chain is pretty stable by increasing the amount of training data. This outcome suggests that both could be candidate methods for the analysis of very large data sets. However, a tiny feature selection is always demanded by Markov chain before learning the model to reduce dimensionality and let the method be applicable to new data sets.

8.2.4 Cross-domain analysis

The second experiment is about cross-domain sentiment classification. The goal is to assess whether paragraph vector is able to bridge the semantic gap between the source and target domains, despite no explicit transfer learning mechanism. For this purpose, paragraph vector

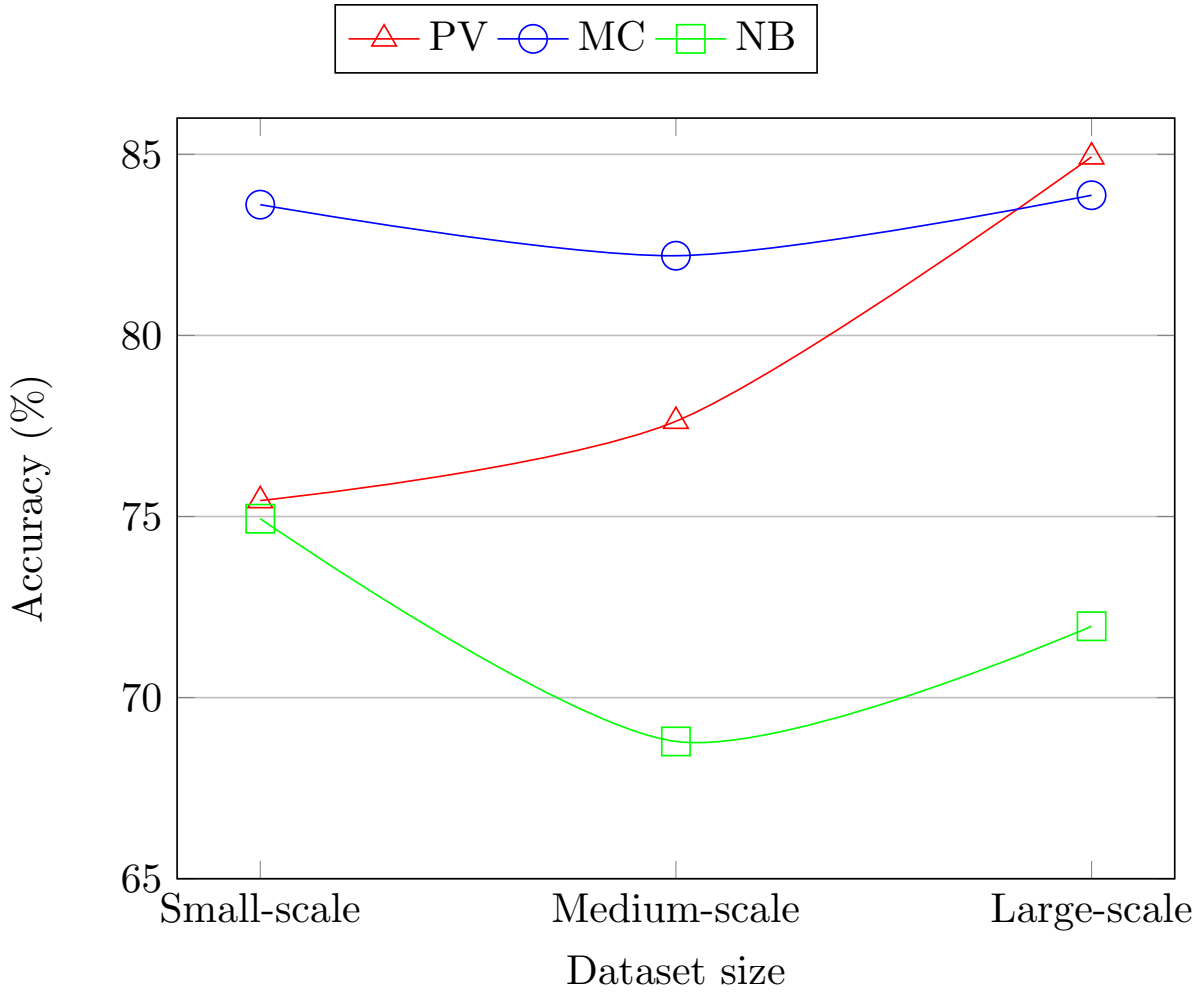


Figure 8.4: Average in-domain accuracy of the compared methods, varying the dataset size. *Source: [DMPP17b].*

has been compared with the Markov chain method in the 12 source-target configurations of the domains considered, namely B → E, B → M, B → J, E → B, E → M, E → J, M → B, M → E, M → J, J → B, J → E, J → M. As in the previous experiment, Naïve Bayes is used as a baseline. The results are shown in Table 8.2 and in Figure 8.5.

	1.6k-0.4k			16k-4k			80k-20k		
Domain(s)	PV	MC	NB	PV	MC	NB	PV	MC	NB
Cross-domain experiments (*source* → *target*)									
B → E	70.75%	69.29%	67.50%	67.27%	71.22%	55.85%	73.24%	74.05%	58.77%
B → M	66.75%	70.85%	70.50%	80.25%	79.32%	61.15%	81.97%	79.01%	61.99%
B → J	73.25%	79.70%	63.75%	70.60%	71.83%	53.38%	74.87%	75.99%	54.92%
E → B	74.00%	54.00%	64.50%	78.80%	80.10%	65.43%	76.87%	79.19%	66.15%
E → M	71.50%	56.75%	70.75%	76.17%	76.20%	64.43%	76.86%	77.15%	66.06%
E → J	82.75%	74.25%	72.00%	79.47%	80.49%	63.23%	80.80%	81.91%	73.09%
M → B	74.75%	65.75%	65.25%	85.55%	86.05%	76.55%	85.21%	83.81%	69.05%
M → E	71.75%	68.18%	65.25%	75.32%	77.10%	66.35%	74.79%	72.87%	63.94%
M → J	82.25%	81.95%	63.25%	73.45%	74.86%	62.03%	76.96%	78.58%	67.26%
J → B	66.25%	75.25%	62.50%	69.62%	80.55%	64.48%	76.53%	78.55%	65.88%
J → E	76.50%	80.60%	75.75%	78.55%	79.76%	68.90%	80.08%	81.79%	70.88%
J → M	74.25%	81.25%	72.50%	70.77%	74.30%	63.25%	76.07%	77.93%	66.27%
Average	**73.73%**	**71.49%**	**67.79%**	**75.49%**	**77.65%**	**63.75%**	**77.85%**	**78.40%**	**65.36%**

Table 8.2: Comparison between paragraph vector and Markov chain in cross-domain sentiment classification, using Naïve Bayes as a baseline. Nk-Mk means that the experiment has been performed by using N*1000 instances as the training set and M*1000 instances as the test set. X → Y means that the model has been learned on reviews from the source domain X and then applied to reviews from the target domain Y.

As expected, the accuracy of the baseline algorithm is very low on average, because Naïve Bayes has no transfer learning capability. Markov chain performs much better than Naïve Bayes on average, thanks to the transfer learning mechanism described in Section 7.2 and summarized in Section 8.2.1. Differently from the in-domain problem, where the accuracy achieved by Markov chain is stable with respect to the dataset size, here its accuracy improves by increasing the number of training examples, as shown by Figure 8.5. This means that Markov chain requires large amount of training data in order to effectively transfer the knowledge extracted from the source domain to the target domain.

The most surprising outcome is certainly the accuracy obtained by paragraph vector, which is comparable with Markov chain on average and even higher in the analysis of the smallest data sets. This outcome is quite astonishing if we think that paragraph vector does not provide for a transfer learning phase. It could be explained by considering that paragraph vector is instrinsically able to handle the word ordering, since each document is represented by a dense vector that is trained to predict the following word within the document. During training, terms

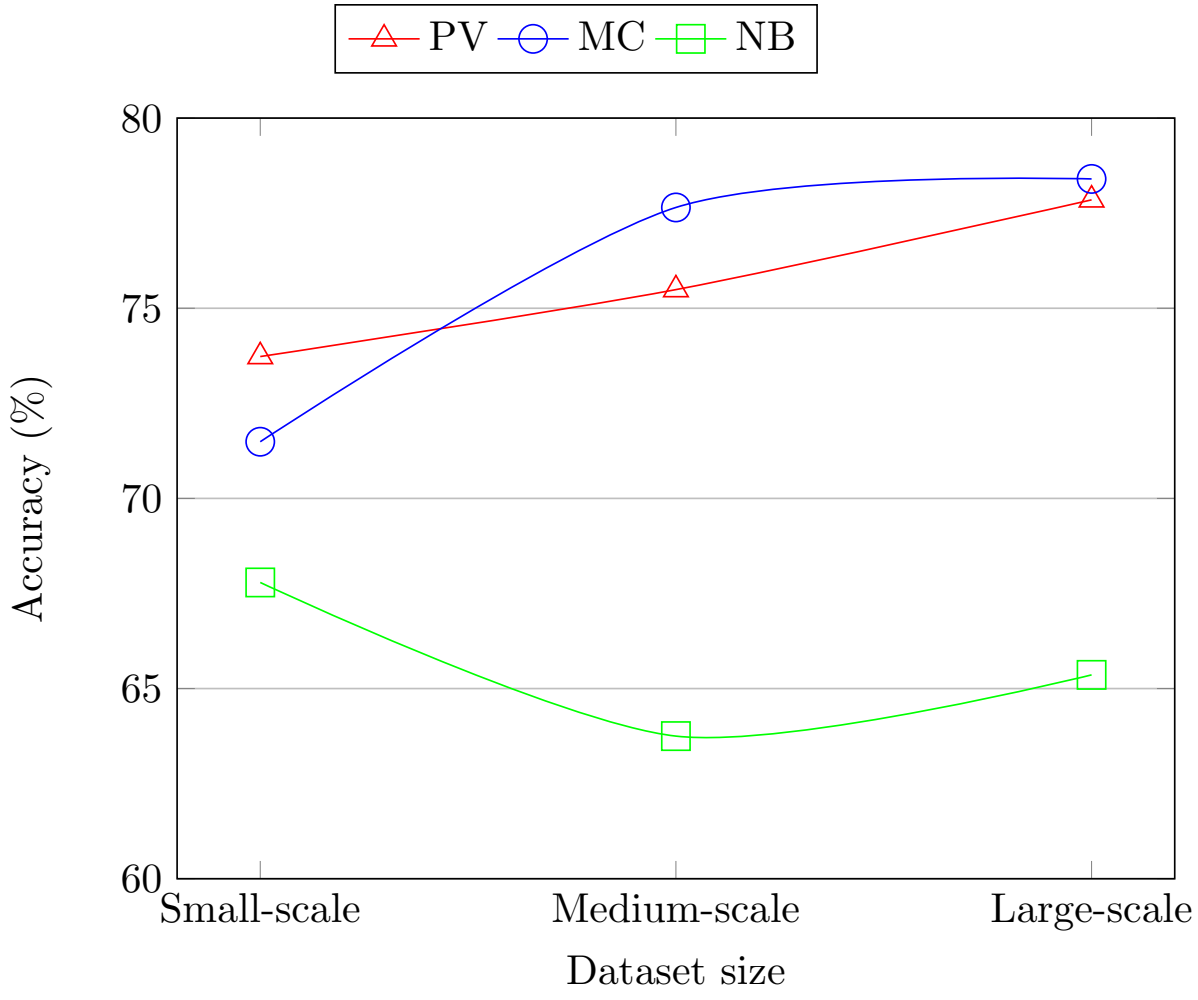


Figure 8.5: Average cross-domain accuracy of the compared methods, varying the dataset size. *Source: [DMPP17b].*

are mapped into a vector space where distance is related to word semantics. For example, the distance between *good* and *robust* is less than the distance between either of such terms and *town*. The fact that paragraph vector learns vector representations without any kind of supervision is probably what could have helped more bridging the inter-domain semantic gap.

One could object that the average cross-domain accuracy in the smallest data sets is surprising. In fact, while paragraph vector requires big training sets to perform well in in-domain tasks (Section 8.2.3), it outperforms Markov chain on average in the respective cross-domain configurations, where the two methods achieve 73.73% and 71.49% accuracy respectively. The rationale of this outcome has to be found in the concept of supervision. Markov chain includes a transfer learning mechanism, which relies on labeled data to transfer semantics from the source domain to the target domain. As a consequence, when few labeled data are available, the algorithm does not achieve high accuracy on target domains. On the other hand, paragraph vector does not explicitly handle transfer learning and relies on an unsupervised approach to map terms in a vector space, i.e. the feature space. For such a reason, paragraph vector is less affected by the change of domain than Markov chain.

This experiment has shown that the ability of paragraph vector to generalize and extract word semantics could foster cross-domain sentiment classification. This is achieved thanks to unsupervised pre-training, which has been used to learn fixed-length vector representations of terms. Unsupervised pre-training of deep learning algorithms, if opportunely combined with a proper transfer learning approach, can be a breakthrough of ad hoc cross-domain sentiment solutions in big data scenarios.

8.2.5 Multi-source training

To support the claim that deep learning and transfer learning solutions can, if combined, break through cross-domain sentiment classification, a third experiment is performed, which evaluates the impact of a multi-source approach on the transfer learning capability of paragraph vector. Multi-source basically means that 3 out of 4 domains are used to train the model, which is then tested on the remaining domain. For instance, the model is built on Books, Electronics and Movies, and then applied to Jewelry. Such a configuration is referred as $* \rightarrow$ J, and the others are indicated analogously. This still is a cross-domain sentiment classification problem, because the model is learned on labeled data from some domains but its performance is evaluated on a different unlabeled domain. The only difference between the single-source paragraph vector (ss-PV) and the multi-source paragraph vector (ms-PV) is that the latter relies on heterogeneous data sources when training the model.

The rationale of such an experiment is that, training the model on heterogeneous domains, more variability in instances can be captured and paragraph vector could automatically learn how to handle the heterogeneity of language, enhancing its transfer learning capability and, as a consequence, its cross-domain performance.

Table 8.3 and Figure 8.6 report the comparison between the single-source paragraph vector and the multi-source paragraph vector in cross-domain sentiment classification. The training on

	1.6k-0.4k		16k-4k		80k-20k	
Domain(s)	ss-PV	ms-PV	ss-PV	ms-PV	ss-PV	ms-PV
Multi-source experiments						
* → B	71.67%	76.85%	77.99%	78.22%	79.54%	81.38%
* → E	73.00%	75.08%	73.71%	78.04%	76.04%	78.46%
* → J	79.42%	75.15%	74.51%	78.79%	77.54%	81.05%
* → M	70.83%	76.10%	75.73%	80.73%	78.30%	82.06%
Average	**73.73%**	**75.80%**	**75.49%**	**78.95%**	**77.85%**	**80.74%**

Table 8.3: Comparison between the single-source paragraph vector (referred as ss-PV) and the multi-source paragraph vector (referred as ms-PV) in cross-domain sentiment classification. * → Y means that the model has been applied to reviews from the target domain Y, after learning on the others except from Y. Nk-Mk means that the experiment has been performed by using N*1000 source instances as the training set and M*1000 target instances as the test set. Actually, the number of training instances is such that there is the same number of examples from each of the source domains.

multiple heterogeneous domains allows paragraph vector learning better semantic-bearing word representations than using a single domain only. It is pretty easy to see that ms-PV outperform ss-PV on average, achieving accuracy from 2% to 3% higher independently of the dataset size. This proves that even a simple gimmick as the multi-source approach is effective to increase the accuracy of paragraph vector in cross-domain tasks, despite it has not been designed to explicitly handle transfer learning. Furthermore, the outcome supports the claim that combining deep learning techniques as paragraph vector with more advanced transfer learning solutions could break through cross-domain sentiment classification.

8.2.6 Final remarks

This section has shown to what extent methods to learn distributed text representation without supervision can support cross-domain sentiment classification, enhancing transfer learning so as to bridge the semantic gap between heterogeneous domains.

The outcome of such an investigation shows that paragraph vector needs very large training sets to learn accurate word and paragraph representations. Once enough data are available to train the model, it achieves accuracy comparable with the Markov chain method introduced in Chapter 7. This not only occurs in in-domain tasks but also in cross-domain experiments, although no explicit transfer learning mechanism is provided by paragraph vector.

An approach is finally proposed, where a model is learned on multiple source domains, then applied to a different target domain. The multi-source paragraph vector outperforms the single-source paragraph vector, suggesting that transfer learning can be fostered by capturing more variability in instances. Such preliminary results suggest that a combination of deep learning and transfer learning techniques could bring to a breakthrough of ad hoc cross-domain

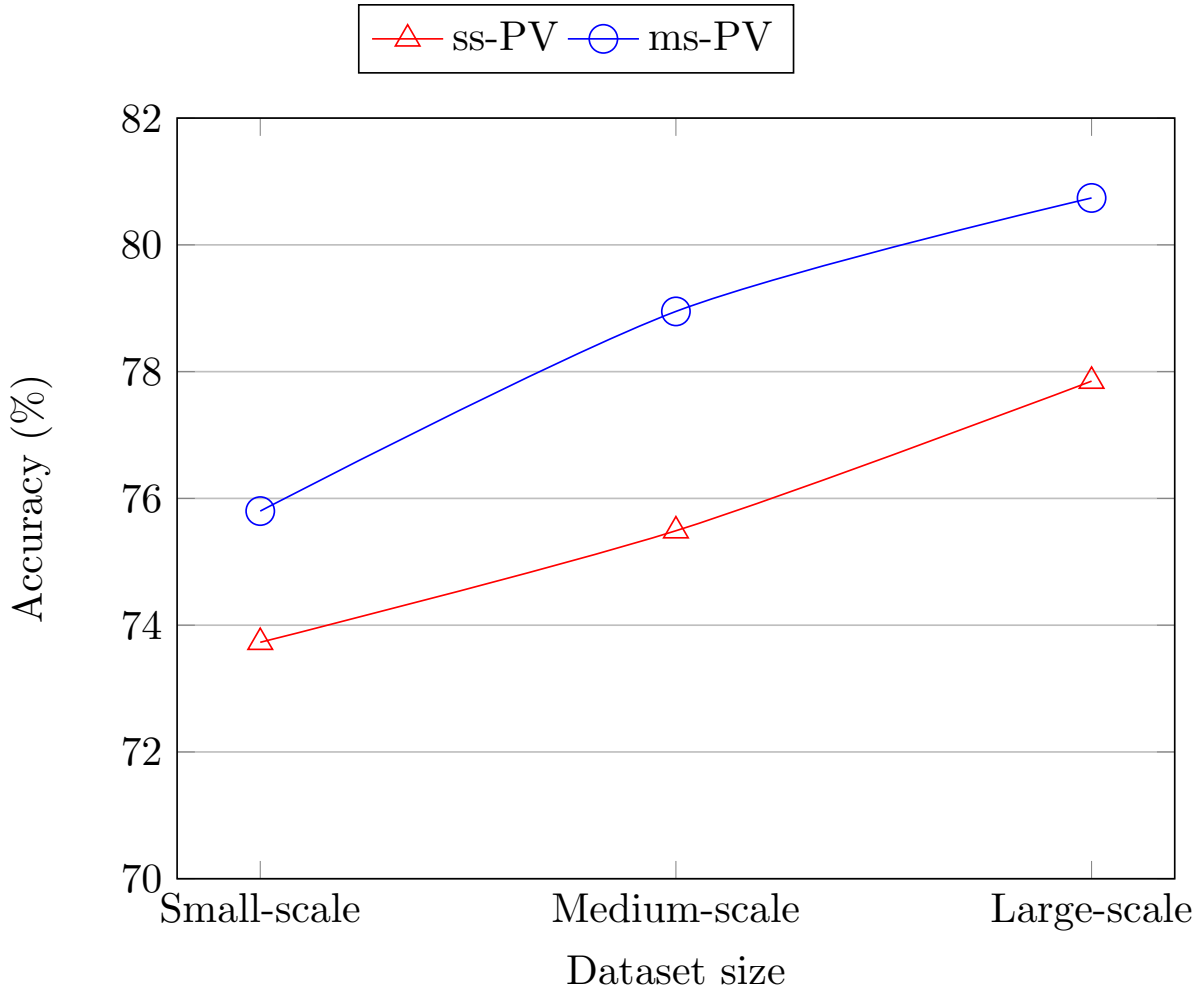


Figure 8.6: Average cross-domain accuracy of the compared methods, when training on multiple source domains, and varying the dataset size. *Source: [DMPP17b].*

solutions in big data scenarios.

8.3 Fine-tuning of memory-based deep neural networks for transfer learning

In this section, the investigation started in Section 8.2 is expanded, assessing whether memory-based neural networks are beneficial for cross-domain tasks. In particular, Gated Recurrent Unit (GRU) is added to the previous analysis, where distributed text representation techniques as paragraph vector have been compared with ad hoc transfer learning and cross-domain solutions as the Markov chain method introduced in Section 7.2. Furthermore, fine-tuning on target instances is also examined to evaluate to what extent it can enhance the transfer learning ability of memory-based deep neural networks.

Readers can find the main features of paragraph vector and Markov chain in Section 8.2.1, whereas a brief description of gated recurrent unit will be presented below.

8.3.1 Gated recurrent unit

Gated Recurrent Unit (GRU), proposed by Cho et al. [CVMG+14], is an evolution of Long Short-Term Memory (LSTM), presented by Hochreiter and Schmidhuber [HS97]. Long short-term memory is a deep architecture that has been introduced to overcome the vanishing (or blowing up) gradient problem [Hoc98] that affects recurrent nets when signals are backpropagated through long time sequences. Indeed, long short-term memory can learn to bridge time intervals in excess of $1,000$ discrete time steps without loss of short time lag capabilities, by enforcing constant error flow through internal states of special units. LSTM units are memory cells composed of different gates, namely input gate, output gate, and forget gate. The input gate of a unit u allows protecting the memory contents stored in j from perturbation by irrelevant inputs. The output gate of u allows protecting other units from perturbation by irrelevant memory contents stored in j. The forget gate of u allows forgetting the memory contents that are no longer relevant. A LSTM unit is able to decide whether to keep the existing memory content via the gates. If a relevant feature from an input sequence is detected, the unit is able to preserve this information over a long distance. This property makes long short-term memory suitable for capturing long-distance dependencies.

Gated recurrent unit extends long short-term memory by making each recurrent unit adaptively capture dependencies of different time scales. A gated recurrent unit is similar to the LSTM unit, but it only presents two gates, as shown in Figure 8.7. The activation of a gated recurrent unit is ruled by an update gate, which controls how much information from the previous hidden state will carry over to the current hidden state. A reset gate allows the hidden state to drop any information that is found to be irrelevant later on in the future. As each hidden unit has separate reset and update gates, it will learn to capture dependencies over different time scales.

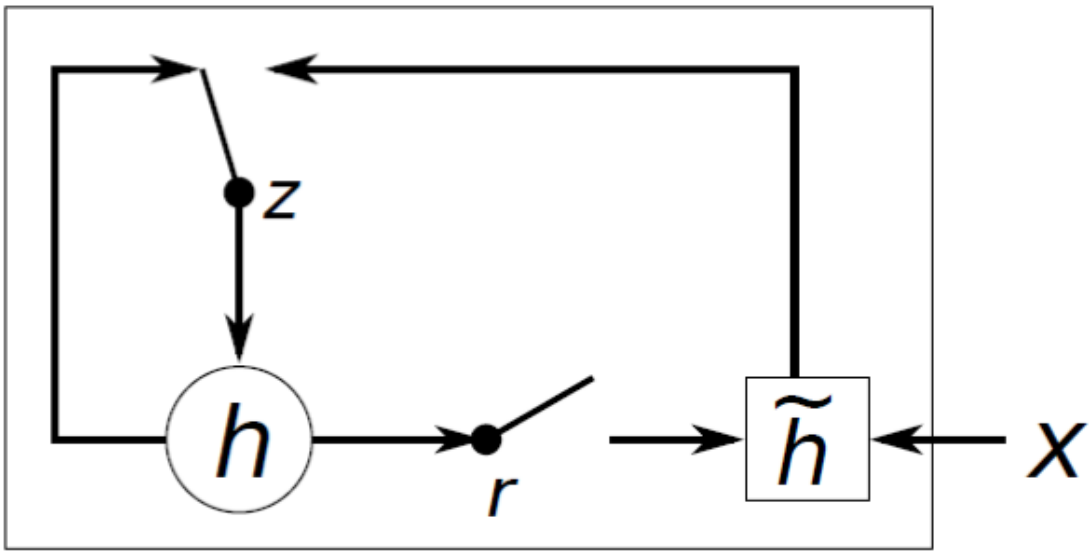


Figure 8.7: An illustration of a GRU. The update gate z selects whether the hidden state is to be updated with a new hidden state $\widetilde{h}$. The reset gate r decides whether the hidden state is ignored. *Source: [CVMG+14].*

LSTM based schemes have already been proved to work well for sentiment classification [TQL15]. In this section, gated recurrent units are applied to cross-domain sentiment classification, to assess whether they are able to automatically bridge the semantic gap between the source and the target domains. Alike paragraph vector, gated recurrent unit is a deep architecture and does not rely on a transfer learning mechanism. However, GRU gates, which allow each unit working as a memory wherein relevant information can be stored and preserved, make gated recurrent units suitable for cross-domain problems. Important domain-independent information can be automatically extracted by gated recurrent units if trained with appropriate amount of data.

8.3.2 In-domain analysis

The experimental protocol is the one described in Section 8.2.2. In-domain results act as a baseline for the cross-domain analysis. Table 8.4 shows the results over the 4 domains of Amazon reviews corpus used for the previous investigation, namely Books (B), Movies (M), Electronics (E) and Clothing-Shoes-Jewelry (J).

	1.6k-0.4k			16k-4k			80k-20k		
Domain(s)	PV	MC	GRU	PV	MC	GRU	PV	MC	GRU
In-domain experiments									
B → B	67.25%	79.25%	83.50%	75.40%	81.90%	73.75%	84.74%	83.84%	89.63%
M → M	79.75%	91.23%	76.75%	74.87%	82.43%	78.92%	84.11%	80.23%	79.50%
E → E	79.25%	92.00%	80.75%	80.15%	80.72%	82.50%	85.61%	84.41%	85.24%
J → J	75.50%	71.97%	82.25%	80.08%	83.76%	85.00%	85.25%	86.98%	84.40%
Average	**75.44%**	**83.61%**	**80.81%**	**77.63%**	**82.20%**	**80.04%**	**84.93%**	**83.87%**	**84.69%**

Table 8.4: In-domain comparison among the three techniques used in the analysis. Nk-Mk means that the experiment has been performed by using N*1000 instances as the training set and M*1000 instances as the test set. X → X means that the model has been learned on reviews from a domain X and then applied to different reviews from the same domain.

Gated recurrent unit achieves performance comparable with the other techniques. The outcome is not surprising for some reasons. Firstly, GRU is a recurrent architecture, suitable for modeling sequences of terms. Secondly, GRU is able to capture dependencies of different time scales. Relationships among terms arise independently of how much they are distant. Finally, GRU can store relevant information through time, working as a memory. Readers can find further discussion on paragraph vector and Markov chain in Section 8.2.3.

8.3.3 Cross-domain analysis

The following experiment aims to compare the performance of gated recurrent unit with paragraph vector and Markov chain for cross-domain sentiment classification. The goal is to assess

whether the memory mechanism of gated recurrent units, which allows preserving relevant information through time, makes them suitable for cross-domain learning.

The analysis involves all the source-target configurations of the four domains considered. The detailed results are shown in Table 8.5, whereas the average trend across domains is represented in Figure 8.8.

	1.6k-0.4k			16k-4k			80k-20k		
Domain(s)	PV	MC	GRU	PV	MC	GRU	PV	MC	GRU
Cross-domain experiments (*source* → *target*)									
B → E	70.75%	69.29%	64.00%	67.27%	71.22%	73.15%	73.24%	74.05%	78.20%
B → M	66.75%	70.85%	65.00%	80.25%	79.32%	77.75%	81.97%	79.01%	83.26%
B → J	73.25%	79.70%	69.50%	70.60%	71.83%	77.95%	74.87%	75.99%	82.69%
E → B	74.00%	54.00%	65.75%	78.80%	80.10%	69.50%	76.87%	79.19%	77.00%
E → M	71.50%	56.75%	64.70%	76.17%	76.20%	73.35%	76.86%	77.15%	79.28%
E → J	82.75%	74.25%	69.15%	79.47%	80.49%	81.95%	80.80%	81.91%	85.70%
M → B	74.75%	65.75%	59.00%	85.55%	86.05%	71.65%	85.21%	83.81%	81.21%
M → E	71.75%	68.18%	69.50%	75.32%	77.10%	74.70%	74.79%	72.87%	79.47%
M → J	82.25%	81.95%	68.00%	73.45%	74.86%	76.95%	76.96%	78.58%	82.32%
J → B	66.25%	75.25%	60.50%	69.62%	80.55%	68.20%	76.53%	78.55%	77.19%
J → E	76.50%	80.60%	77.00%	78.55%	79.76%	78.35%	80.08%	81.79%	82.19%
J → M	74.25%	81.25%	63.50%	70.77%	74.30%	72.70%	76.07%	77.93%	77.60%
Average	**73.73%**	**71.49%**	**66.30%**	**75.49%**	**77.65%**	**74.68%**	**77.85%**	**78.40%**	**80.51%**

Table 8.5: Comparison between gated recurrent unit, paragraph vector and Markov chain in cross-domain sentiment classification. Nk-Mk means that the experiment has been performed by using N*1000 instances as the training set and M*1000 instances as the test set. X → Y means that the model has been learned on reviews from the source domain X and then applied to reviews from the target domain Y.

The average trend is pretty clear: the more data gated recurrent unit relies on, the more it performs well. Indeed, gated recurrent unit underperforms the other techniques with small-scale data. Increasing the number of training instances, gated recurrent unit experienced a dramatic growth in accuracy, becoming comparable with both paragraph vector and Markov chain with medium-scale data, and even outperforming them with large-scale data. A reasonable explanation for this behaviour is that its memory mechanism needs appropriate amount of data in order to learn what is actually relevant within a review.

One might argue that, looking at in-domain results in Table 8.4, gated recurrent unit achieves good performance even with small data sets. This means that gated recurrent unit needs few data to capture intra-domain term relationships, whereas few facts are not enough to capture inter-domain dependencies in absence of some explicit transfer learning mechanism. This is reasonable. Just think that identifying the polarity-bearing terms of a single domain could be enough to understand the overall sentiment orientation of a review, whereas the same does

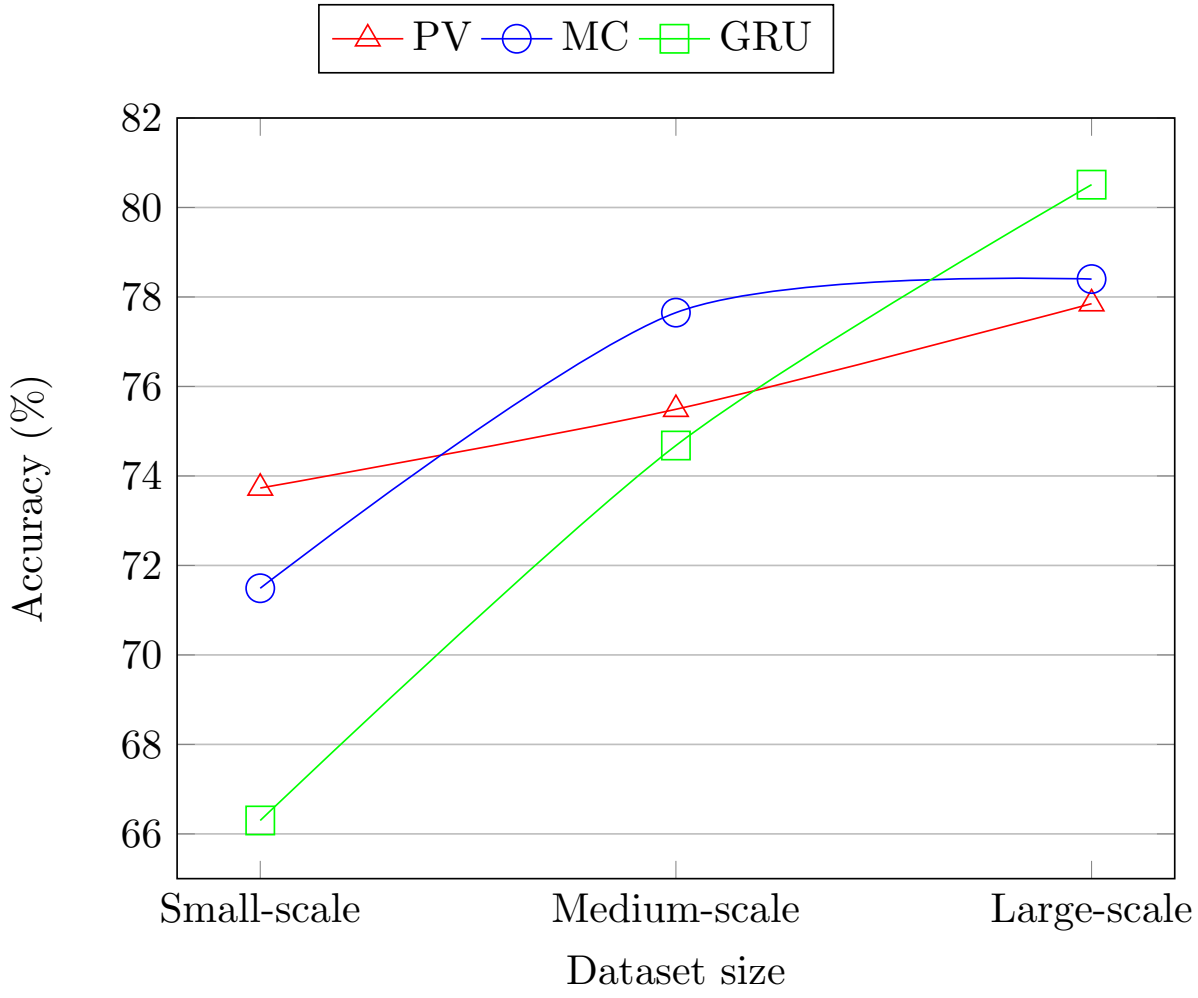


Figure 8.8: Average cross-domain accuracy of the compared methods, varying the dataset size. *Source: [PMPD].*

not hold between distinct domains, because of language heterogeneity. The polarity-bearing terms of the source domain generally differ from those of the target domain. In order to support the knowledge transfer from source to target, the identification of relevant hidden concepts is essential, more than finding important terms. Careful readers could argue that paragraph vector, similarly to gated recurrent unit, does not provide a transfer learning phase, achieving good performance with small-scale data anyway. This is true, but it should not be forgotten that paragraph vector is able to capture word semantics without supervision [LM14]. This feature makes paragraph vector suitable for bridging the inter-domain semantic gap, as shown in Section 8.2.4.

The outcome of cross-domain experiments suggests that gated recurrent units are automatically able to decide which information is better to preserve even across heterogeneous domains.

However, gated recurrent unit needs a large-scale training set in order to perform well in cross-domain tasks. Since gated recurrent unit does not rely on explicit transfer learning mechanisms, it requires more data to extract hidden relevant concepts to bridge the semantic gap between distinct domains.

8.3.4 Fine-tuning for transfer learning and cross-domain

The experiments described below assess whether fine-tuning on target instances effectively supports deep learning techniques in cross-domain sentiment classification.

Fine-tuning consists in using a labeled sample of target instances to refine a model previously learned on a source domain. The sample should be reasonably small for both theoretical and practical reasons. From a theoretical point of view, if the sample used for fine-tuning is too large, the cross-domain task is converted into an in-domain problem. In such a case, where appropriate amount of labeled target instances is available, cross-domain would be no longer needed. An in-domain model could easily be learned in that case. On the other hand, readers already know that cross-domain learning is essential from a practical point of view, since most real world data are unlabeled. Finding a large labeled sample is challenging in practice, and manually labeling it is even infeasible. Therefore, using a large sample is not a viable alternative for tuning a pre-trained model. Only if the sample is small, categorization by human experts becomes a good option to increase cross-domain efficacy.

Beyond being a good trade-off solution between its cost and the improvement of performance that it guarantees, fine-tuning could be even critical for techniques that do not rely on explicit transfer learning mechanisms. In particular, here we assess whether fine-tuning of deep neural networks can bring to an improvement of ad hoc cross-domain solutions. For this purpose, 250 and 500 target samples have been used. The detailed cross-domain results with fine-tuning are shown in Table 8.6, whereas the average trend is plotted in Figure 8.9 and compared with the accuracy that paragraph vector and gated recurrent unit obtained without tuning on target instances.

As can be seen, paragraph vector is almost unaffected by fine-tuning, regardless of the dataset size. This behaviour is explained by the ability of paragraph vector to capture word semantics without supervision, so it does not take advantage of further target instances. Paragraph vector automatically handles the heterogeneity of language by discovering hidden relationships between semantically similar words [LM14]. On the other hand, the benefits of fine-tuning dramatically affect gated recurrent unit, which is not inherently able to align domains without supervision. The improvement is particularly evident with small-scale data, and decreases by growing the amount of source data employed to pre-train the model. The reason is pretty obvious. When few training data are available, gated recurrent unit cannot capture inter-domain dependencies, and even a small sample of target data leads to a significant boost of performance. The impact is a bit reduced with medium-scale data, mainly for two factors. The first factor is the increased capability of gated recurrent unit in bridging the inter-domain semantic gap without fine-tuning, as already shown in Section 8.3.3. The second factor is that 250 and 500 instances are two orders

	1.6k-0.4k		16k-4k		80k-20k	
Domain(s)	PV	GRU	PV	GRU	PV	GRU
Fine-tuning with 250 instances (*source* → *target*)						
B → E	70.56%	69.00%	73.86%	74.85%	72.40%	79.05%
B → M	67.78%	68.50%	76.33%	79.45%	83.83%	84.51%
B → J	76.67%	71.50%	71.03%	79.80%	75.16%	82.03%
E → B	63.89%	68.50%	78.61%	70.20%	74.73%	77.83%
E → M	73.78%	67.00%	65.28%	73.85%	79.18%	80.17%
E → J	82.22%	71.00%	79.14%	84.60%	81.14%	86.44%
M → B	79.17%	67.50%	80.08%	72.95%	81.22%	81.22%
M → E	73.89%	72.50%	77.33%	77.10%	74.38%	79.70%
M → J	82.50%	75.50%	75.58%	78.85%	77.92%	82.83%
J → B	64.17%	67.50%	74.81%	70.30%	75.22%	78.78%
J → E	73.89%	77.50%	83.11%	78.50%	80.70%	82.51%
J → M	70.83%	68.00%	62.53%	75.25%	78.25%	79.19%
Average	**73.28%**	**70.33%**	**74.81%**	**76.31%**	**77.84%**	**81.19%**
Fine-tuning with 500 instances (*source* → *target*)						
B → E	70.72%	70.50%	73.54%	73.95%	72.24%	78.56%
B → M	67.56%	68.00%	76.28%	79.55%	83.66%	84.07%
B → J	76.17%	78.00%	70.53%	80.50%	74.89%	82.30%
E → B	66.83%	69.00%	78.21%	71.85%	74.97%	77.39%
E → M	73.03%	72.50%	64.99%	76.65%	79.23%	80.82%
E → J	81.33%	76.50%	79.02%	82.15%	81.15%	85.19%
M → B	79.33%	69.00%	80.94%	76.55%	81.35%	81.78%
M → E	74.61%	72.50%	77.55%	77.25%	74.34%	79.91%
M → J	84.17%	77.50%	74.65%	79.25%	77.70%	82.33%
J → B	66.06%	70.00%	74.42%	71.85%	75.62%	78.52%
J → E	74.83%	78.00%	82.84%	79.55%	80.28%	83.66%
J → M	71.66%	68.50%	62.33%	76.60%	78.04%	79.59%
Average	**73.86%**	**72.50%**	**74.61%**	**77.14%**	**77.79%**	**81.18%**

Table 8.6: Comparison between gated recurrent unit, paragraph vector and Markov chain in cross-domain sentiment classification with fine-tuning on a small set of labeled target instances. Nk-Mk means that the experiment has been performed by using N*1000 instances as the training set and M*1000 instances as the test set. X → Y means that the model has been learned on reviews from the source domain X and then applied to reviews from the target domain Y. 250 and 500 target instances have been used for tuning.

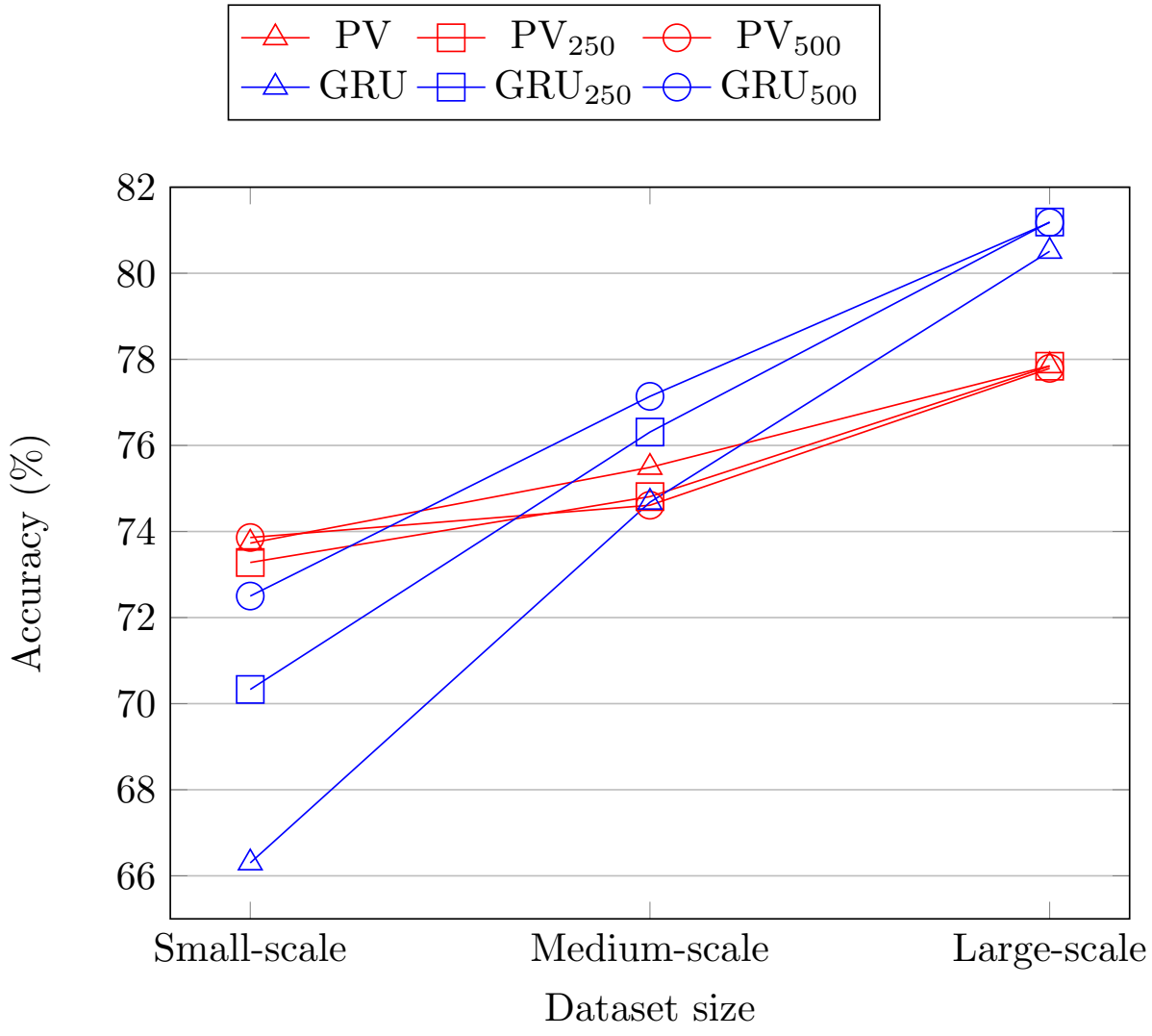


Figure 8.9: Average accuracy achieved by the compared methods when fine-tuning on small samples of target instances is performed to foster cross-domain sentiment classification. The subscripts 250 and 500 are referred to the number of instances sampled from the target domain in order to perform fine-tuning. *Source: [PMPD].*

of magnitude less than the dataset size considered, whereas the small-scale data were just one order of magnitude more than the amount used for tuning. It is obviously challenging to increase the performance of a model pre-trained on a set of medium-scale data, by using only such a small sample for tuning. The same two factors also affect performance improvement when large-scale data are taken into account.

In spite of the few instances used, fine-tuning is beneficial to gated recurrent unit on average with respect to the considered configurations. With small-scale data, the sample of 250 target instances improves accuracy by approximately 4%, whereas doubling the tuning instances, accuracy increases by about 2% more. With medium-scale data, the smaller sample boosts accuracy by about 1.6%, whereas the bigger one by less than 1% with respect to the smaller. Finally, the accuracy increases by less than 1% with respect to the configuration without tuning when large-scale data are considered, independently of the size of the tuning sample.

The outcome of the analysis proves that fine-tuning on a small sample of labeled target data is beneficial to deep architectures that do not have neither explicit transfer learning mechanisms nor the capability to automatically detect semantically similar terms without supervision.

8.3.5 Final remarks

The investigation on deep learning for cross-domain sentiment classification has been carried on by including Gated Recurrent Unit (GRU) [CVMG+14] in the analysis. Gated recurrent unit is a deep architecture, evolution of long short-term memory, able to adaptively capture dependencies of different time scales. Similarly to paragraph vector, it does not provide any explicit transfer learning mechanism. Binary sentiment classification experiments have shown that gated recurrent unit performs worse than paragraph vector and Markov chain with small and medium-scale data sets, whereas it outperforms both when trained on large-scale data. The outcome suggests that GRU memory units are beneficial for cross-domain, but require large-scale data in order to learn accurate word relationships.

The impact of fine-tuning on transfer learning and cross-domain tasks has also been shown. Fine-tuning is an explicit transfer learning mechanism, where a small sample of target instances is used to tune the parameters of a model learned from the source domain. When tuned with samples of target data, gated recurrent unit achieves accuracy comparable with the other methods, even with few training data, proving that fine-tuning helps transfer learning between heterogeneous domains. The labeled sample of target instances should be small in order for human experts to manually label data without too much effort.

8.4 Combining memory-based deep architectures and distributed text representations for sentiment classification

This section shows massive experiments on recent deep learning advances, pointing out the characteristics that mostly affect the identification of sentiment. In particular, memory-based

deep neural networks are combined with distributed text representations, and their impact on sentiment classification is assessed. Among the memory-based architectures (MemDNNs), Gated Recurrent Unit (GRU) [CVMG+14] and Differentiable Neural Computer (DNC) [GWR+16] are experimented. Their feature weights are initialized by means of global vectors (GloVe) [PSM14], an outstanding technique to learn distributed word representations.

In-domain and cross-domain document-level experiments are performed; the effectiveness of fine-tuning for cross-domain sentiment classification is assessed; document-level experiments are performed on a huge dataset to evaluate to what extent memory-based architectures are scalable with the amount of training data; finally, the suitability of such architectures for single-sentence sentiment classification is assessed. As far as we know, only gated recurrent unit has recently been applied to cross-domain sentiment classification, having been used in combination with word embeddings for Chinese corpora [DHZ+17].

As previously stated, three techniques are taken into account by this analysis, namely gated recurrent unit, differentiable neural computer and global vectors for word representation. The first two are memory-based deep neural networks, whereas the last is an unsupervised approach to learn distributed text representations. Readers can find a description of gated recurrent unit in Section 8.3.1.

8.4.1 Differentiable neural computer

Differentiable Neural Computer (DNC), introduced in [GWR+16] as an evolution of Neural Turing Machines (NTM) [GWD14], is one of the most innovative MemDNN techniques. Differently from previous MemDNN architectures (e.g. GRU), where the memory mechanism was internal to the network, differentiable neural computer uses an external memory to represent and manipulate complex data structures. The neural network can selectively address the external memory, both to read from and write to it, allowing iterative modification of memory content. This makes differentiable neural computer able to learn complex tasks straight away from data, such as finding the shortest path or inferring the missing links in graphs, and answering synthetic questions designed to emulate reasoning in natural language. Figure 8.10 shows the basic behaviour of a differentiable neural computer. It uses differentiable attention mechanisms to define weightings, which represent the degree to which each memory location is involved in a read or write operation. The functional units that determine and apply the weightings are called read and write heads.

In the work it was presented [GWR+16], differentiable neural computer was only applied to small-scale tasks. However, the authors pointed out that differentiable neural computer should be able to seamlessly acquire knowledge and take advantage of exposure to large data sources. This consideration, along with the ability of memory mechanisms to capture inter-domain relationships as shown in Section 8.3.3, makes differentiable neural computer suitable for cross-domain sentiment classification.

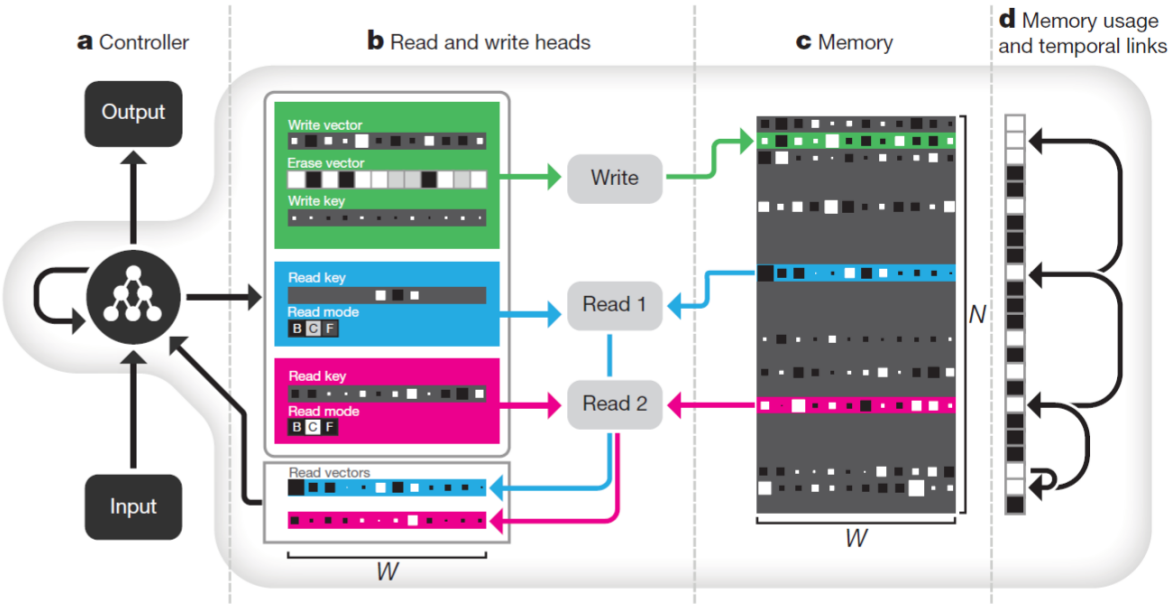


Figure 8.10: In a differentiable neural computer, a recurrent controller network receives input from an external data source and produces output. The controller also outputs vectors that parameterize one write head (green) and multiple read heads (blue and pink). The write head defines a write and an erase vector that are used to edit the memory matrix. Additionally, a write key is used for content lookup to find previously written locations to edit. The read heads can use gates called read modes to switch between content lookup using a read key ('C') and reading out locations either forwards ('F') or backwards ('B') in the order they were written. The memory usage vector records which locations have been used so far, and a temporal link matrix records the order in which locations were written. *Source: [GWR+16].*

8.4.2 Global vectors

Global Vectors (GloVe) is a log-bilinear regression model that have been proposed by Pennington et al. [PSM14] to learn distributed word representation. Alike other methods for learning vector space representation of words, GloVe is able to capture fine-grained syntactic and semantic regularities in an unsupervised fashion, just using vector arithmetic, and solves the data sparsity problem of dense bag-of-words models. GloVe combines the advantages of global matrix factorization and local context window methods: as the former, it efficiently leverages statistical information by training only on the nonzero elements in a word-word co-occurrence matrix; as the latter, it achieves great performance on word analogy, similarity and named entity recognition tasks.

Since unsupervised information extracted by means of distributed word representation fosters the alignment of heterogeneous domains, as shown in Section 8.2, GloVe is a candidate technique for the initialization of the feature weights used by MemDNN architectures.

8.4.3 The impact of labeled training data on memory-based networks

The first experiment checks to what extent the amount of labeled training data affects MemDNN performance. For a matter of comparison with the previous analyses (Section 8.2 and Section 8.3), the same experimental protocol has been carried out. Source-target partitions of three different orders of magnitude have been tested, preserving $80\% - 20\%$ as the source-target ratio, and balancing positive and negative examples. The small-scale dataset has $1,600$ labeled instances as the training set and 400 unlabeled instances as the test set; the medium-scale $16,000$ and $4,000$ respectively; and the large-scale $80,000$ and $20,000$ respectively. Naïve Bayes (NB), Markov chain (MC), Paragraph Vector (PV) and Gated Recurrent Unit with randomly initialized feature weights (GRU) have been considered as baselines.

Figure 8.11 shows the in-domain performance of the various techniques, averaged on the 4 domains considered. As already pointed out, deep learning approaches usually do not perform well when few training data are available. That is the reason why Markov chain outperforms the proposed memory-based techniques with small-scale data. However, gated recurrent unit and differentiable neural computer outperform the other approaches. It deserves attention that gated recurrent unit with feature weights initialized by GloVe (GRU_{GloVe}) achieves higher accuracy with respect to the gated recurrent unit whose features have been initialized with random weights (GRU). Increasing the amount of labeled training data, differentiable neural computer obtains astonishing performance. Its accuracy is 90.08% with medium-scale data, meaning that $16,000$ training examples are enough for the memory mechanism of differentiable neural computer to capture relevant information. Different considerations should be done for gated recurrent unit, whose performance does not increase considering medium-scale data. With large-scale data, the accuracy of differentiable neural computer continues to grow, reaching 91.24%. This outcome makes it interesting to evaluate to what extent the performance of differentiable neural computer can increase. For this purpose, an in-domain test with a huge dataset will be shown later on in

Section 8.4.5. Finally, it may be noted that the performance of gated recurrent unit also improves. A reasonable explanation is that the memory mechanism of gated recurrent unit is automatically able to decide which information is relevant to classification, if trained with large amount of data.

A cross-domain evaluation of the same techniques can be seen in Figure 8.12. The plot displays the accuracy averaged on the 12 source-target configurations of the 4 domains considered. Differentiable neural computer dramatically outperforms all the other techniques regardless of the dataset size. It should be noted that it exceeds by more than 9% the accuracy of Markov chain, which is a non-deep method that was specifically developed to accomplish both transfer learning and sentiment classification. The reason of this outcome resides in several combined factors that lead to semantic comprehension of text. The first factor is the usage of distributed representation to encode text. In particular, GloVe has been employed for text representation and feature weights initialization of both gated recurrent unit and differentiable neural computer. As pointed out by Pennington et al. [PSM14], GloVe combines the advantages of two model families in literature for learning word vectors, namely matrix factorization methods and local context window methods. This means that GloVe also inherits the benefits of paragraph vector, which is able to discover hidden relationships between semantically similar words. The unsupervised information extracted by GloVe aids the alignment of heterogeneous domains. The second factor is the memory mechanism of differentiable neural computer. Once enough training data are available, memory-based deep architectures are automatically able to capture domain-independent information and preserve it in memory through time. The third factor are deep neural networks. In particular, differentiable neural computer is one of the most powerful mechanisms to emulate reasoning and inference problems in natural language. The combined effect of these three factors led to a dramatic improvement of the state of the art in cross-domain sentiment classification. Comparing in-domain and cross-domain results, it could be noted that the accuracy of differentiable neural computer is perfectly aligned by looking at small-scale data, whereas cross-domain performance is slightly worse by increasing the dataset size.

Apart from the astonishing performance of differentiable neural computer, careful readers may have noted the contribution of GloVe representation, looking at gated recurrent unit. As expected, GloVe initialization of the feature weights leads to a substantial increasing of the accuracy with small-scale data, which jumps from 66.30% of GRU to 74.14% of GRU_{GloVe}, about 8%. Comparing in-domain and cross-domain experiments, the combined effect of GloVe representation and memory mechanisms can be seen. The former plays a key role to align heterogeneous domains when few labeled data are available as the training set. The latter is automatically able to extract relevant inter-domain concepts as the amount of labeled training data increases.

8.4.4 Fine-tuning for cross-domain sentiment classification

The second analysis aims at assessing whether fine-tuning is beneficial for memory-based deep neural networks performance. As already pointed out, fine-tuning is the practice of using a labeled sample of target instances to refine a model previously learned on a source domain. The

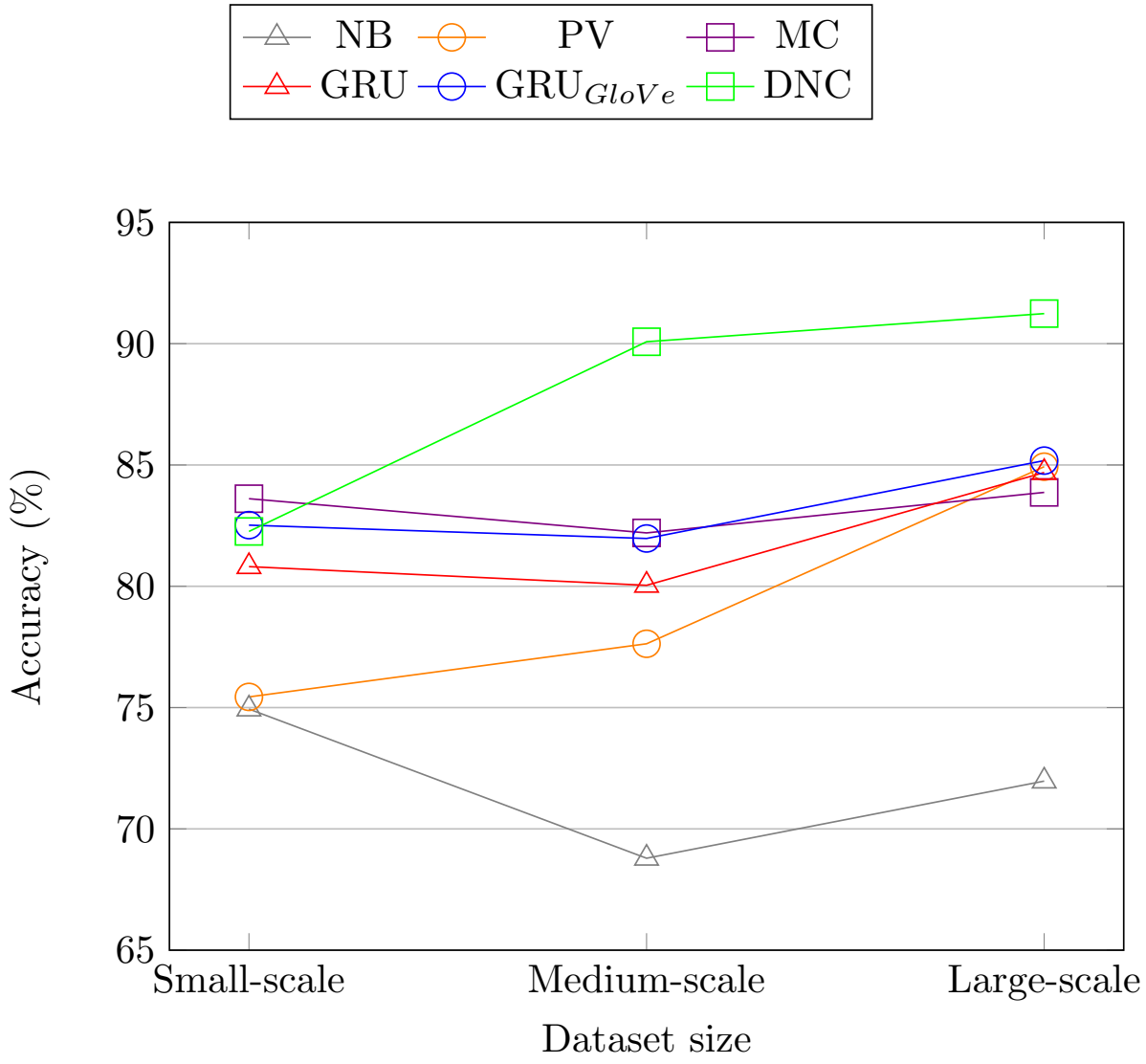


Figure 8.11: Average in-domain accuracy of the compared methods. *Source: [MPPS18].*

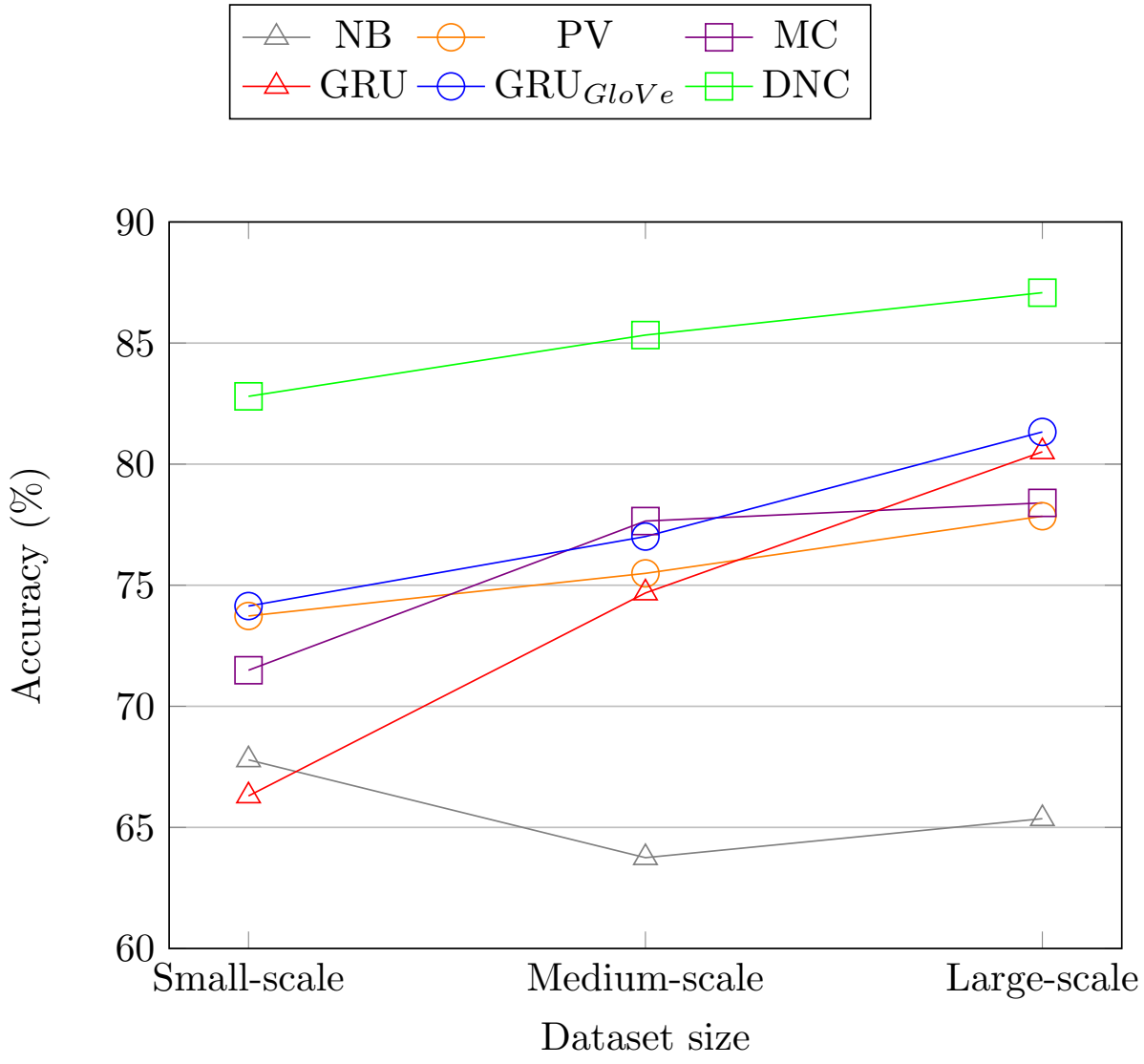


Figure 8.12: Average cross-domain accuracy of the compared methods. *Source: [MPPS18].*

sample is usually small (e.g. hundreds instances) for two main reasons. On one hand, if a large set of labeled instances was available, it would be advisable to learn an in-domain sentiment classifier rather than a cross-domain one. On the other hand, if a large set of labeled instances was not available, the only viable alternative would be to let a team of human experts pre-classify some instances. Manual categorization becomes infeasible as too many instances are required to be labeled. Therefore, fine-tuning on a small sample of labeled target instances generally is a good trade-off between its cost and the expected improvement of performance. To further investigate the performance, fine-tuning has been experimented by using 250 and 500 labeled target samples respectively.

Figure 8.13 shows the effect of fine-tuning on MemDNN architectures. Gated recurrent unit takes a slight advantage of fine-tuning. With reference to small-scale data, accuracy increases from 74.14% to 74.84% using 250 target instances to 75.77% using 500 target instances. It should be noted that GRU performance is much more affected by the GloVe feature weights initialization than by fine-tuning. Indeed, gated recurrent unit with GloVe initialization and no fine-tuning outperforms gated recurrent unit with random initialization and fine-tuning on 500 labeled target instances. As the dataset size increases, the contribution of fine-tuning diminishes, until eventually vanishing with large-scale data. A different behaviour is observed for differentiable neural computer. Fine-tuning only impacts on accuracy when performed on 500 target instances with reference to the small-scale dataset. It is quite obvious that, when few training data are available, even a small sample can considerably affect performance. Gated recurrent unit is a clear evidence of such a behaviour. However, DNC_{250} does not lead to a performance improvement, because differentiable neural computer is a very robust technique, almost unaffected by noise. The attention mechanism to address the external memory makes DNC less sensitive to noise than GRU, whose memory units are internal to the network. As a consequence, differentiable neural computer is less prone than gated recurrent unit to altering memory content. In other words, it is unlikely that differentiable neural computer stores irrelevant information in memory. 250 target instances are not relevant enough for differentiable neural computer, and are considered as noise by the network. The same considerations apply to experiments with medium-scale and large-scale data, where both 250 and 500 target instances do not affect performance.

8.4.5 Large-scale document sentiment classification

The third experiment is an in-domain sentiment classification task on a huge dataset. This analysis let us understand to what extent MemDNN architectures with the GloVe feature weights initialization are suitable as sentiment classifiers in big data scenarios. Here MemDNN architectures are compared with many sentiment classification techniques, in particular several variants of Character-level Convolutional Neural networks (CharCNN), proposed in [ZL15b] and empirically explored in [ZZL15].

For result comparison purposes, two very large datasets have been constructed. The former deals with binary in-domain sentiment classification, where the goal is to distinguish positive

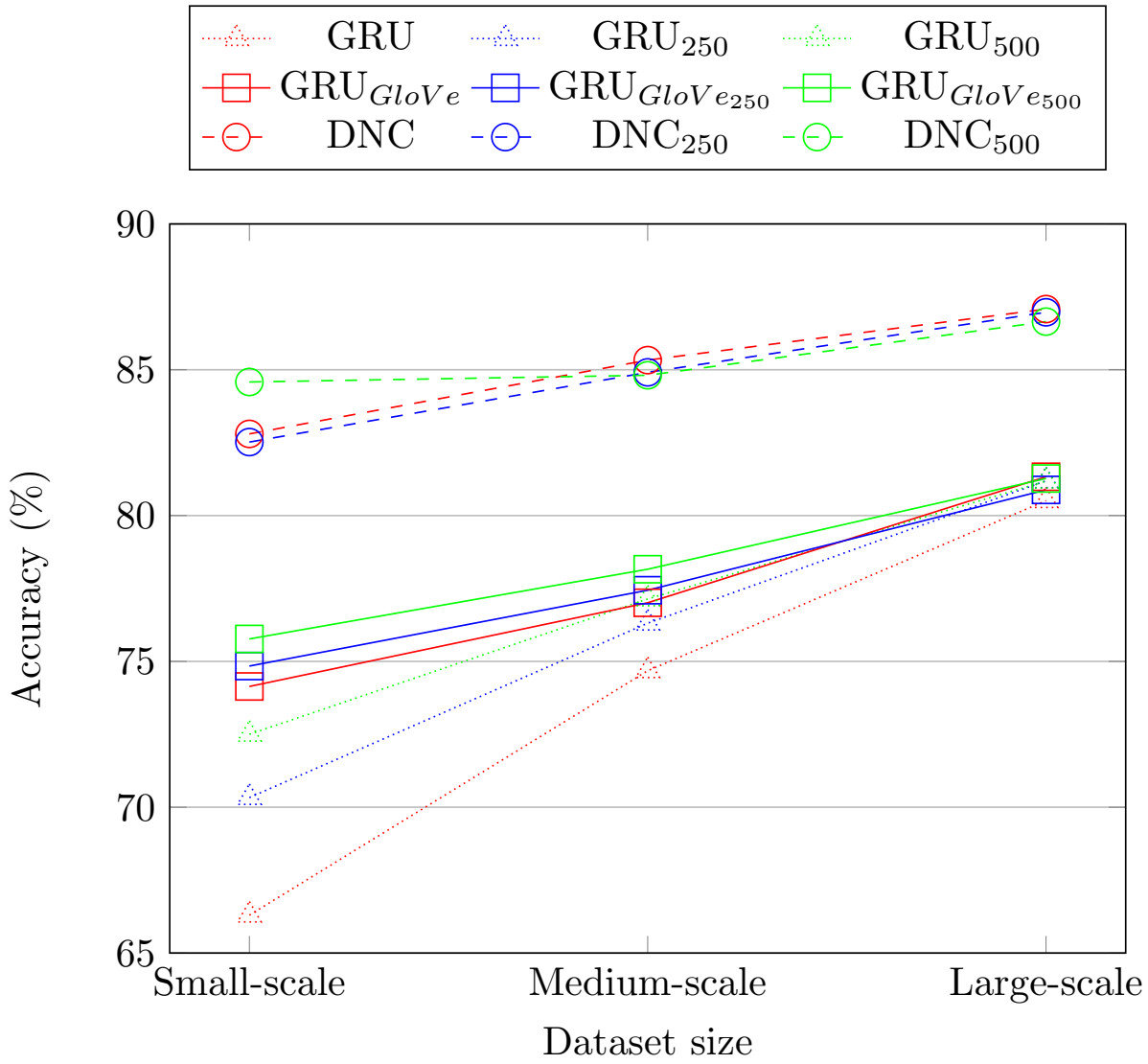


Figure 8.13: Average accuracy achieved by the compared methods when fine-tuning on small samples of target instances is performed to foster cross-domain sentiment classification. The subscripts 250 and 500 represent the number of labeled target instances utilized for fine-funing. *Source: [MPPS18].*

and negative instances. The latter aims at identifying the full score assigned to instances (i.e. from 1 to 5). So that is a fine-grained in-domain sentiment classification task. The binary dataset contains $1,800,000$ training samples and $200,000$ testing samples for each polarity sentiment. The fine-grained contains $600,000$ training samples and $130,000$ testing samples for each of the five classes. In both datasets, samples have been taken in equal proportion from the 4 domains considered. Differently from the previous experiments, review title has also been considered, together with review content.

Model	Binary	Fine-grained
BoW	90.40%	54.64%
BoW Tf-Idf	91.00%	55.26%
n-grams	92.02%	54.27%
n-grams Tf-Idf	91.54%	52.44%
Bag-of-means	81.61%	44.13%
LSTM	93.90%	59.43%
Lg. w2v Conv	94.12%	55.60%
Sm. w2v Conv	94.00%	57.41%
Lg. w2v Conv. Th.	94.20%	56.25%
Sm. w2v Conv. Th.	94.37%	57.50%
Lg. Lk. Conv	94.16%	54.05%
Sm. Lk. Conv	94.15%	56.34%
Lg. Lk. Conv. Th.	94.48%	57.61%
Sm. Lk. Conv. Th.	94.49%	56.81%
Lg. Full. Conv	94.22%	59.11%
Sm. Full. Conv	94.22%	59.12%
Lg. Full. Conv. Th.	94.49%	59.46%
Sm. Full. Conv. Th.	94.34%	59.47%
Lg. Conv	94.49%	58.69%
Sm. Conv	94.50%	59.47%
Lg. Conv. Th.	95.07%	59.55%
Sm. Conv. Th.	94.33%	59.57%
GRU_{GloVe}	94.07%	59.55%
DNC	**95.51**%	**61.45**%

Table 8.7: Accuracy achieved by the compared methods on very large datasets constructed from the Amazon reviews corpus. Binary and fine-grained refer to 2-class and 5-class in-domain sentiment classifications respectively. CharCNN variants are prefixed with Lg. or Sm., as in [ZZL15]. The best accuracy is in bold.

Apart from the several variants of CharCNN, results have also been compared with other methods, including Long Short-Term Memory networks (LSTM), Bag-of-means [LKW15] and some Bag-of-words (BoW) based configurations. Careful readers can find further details

on these methods along with their parameters in [ZZL15]. Table 8.7 shows the accuracy of MemDNN methods and the state-of-the-art techniques. Gated recurrent unit achieves comparable performance with the other methods, being slightly more accurate than long short-term memory. This outcome is not surprising, since gated recurrent unit is an evolution of long short-term memory, but both have a built-in memory mechanism. On the other hand, the accuracy of differentiable neural computer is astounding. It outperforms all the other techniques with reference to both binary and fine-grained configurations. In the fine-grained task, accuracy is almost 2% higher than the previous methods. This outcome is significant in a multinomial classification problem, where predicting the correct class is challenging. To the best of our knowledge, it is the first time that a method achieves accuracy higher than 60% on the Amazon reviews corpus with reference to a fine-grained sentiment classification task.

8.4.6 Single-sentence sentiment classification

While the previous experiments deal with document sentiment classification, the last one focuses on single-sentence sentiment classification. The benchmark corpus used for the analysis is the Stanford Sentiment Treebank (SST), which was introduced by Socher et al. [SPW$^+$13]. The treebank is built on a corpus of movie review excerpts, composed of $11,855$ sentences, half of which are positive and half negative. The sentences have been parsed with the Stanford parser [KM03] into $215,154$ syntactically plausible phrases. Each phrase has been annotated by 3 human experts into 5 possible categories, namely negative, somewhat negative, neutral, somewhat positive and positive. Similarly to the Amazon reviews corpus, neutral phrases are discarded in binary sentiment classification tasks. According to the work by Socher et al. [SPW$^+$13], $8,544$ sentences are used as the training set, $1,101$ as the validation set, and $2,210$ as the test set.

In the last few years, plenty of techniques have been applied to the Stanford sentiment treebank. Socher et al. [SPW$^+$13] presented Recursive Neural Tensor Networks (RecNTN) in the same work they introduced the treebank, and compared their algorithm on the SST with Naïve Bayes with unigram features (NB), Naïve Bayes with unigram and bigram features (BiNB), Support Vector Machine with unigram and bigram features (SVM), Recursive Neural Networks (RNN) [SPH$^+$11] and Matrix-Vector RNN (MV-RNN) [SHMN12]. Kalchbrenner et al. [KGB14] proposed Dynamic Convolutional Neural Network (DCNN), comparing its performance on the Stanford sentiment treebank with Max Time-Delay Neural Networks (Max-TDNN) [CW08], and a Neural Bag-of-Words (NBoW) model. Dos Santos and Gatti [dSG14] introduced Character to Sentence Convolutional Neural Network (CharSCNN) and applied it to the treebank. A variant of CharSCNN has been trained by using word embeddings only (SCNN). Other two variants of the previous, referred as CharSCNN ph. and SCNN ph., have been trained by exploiting phrases representations in addition to sentence representation. Kim [Kim14] experimented some variants of Convolutional Neural Networks (CNN-rand, CNN-static, CNN-non-static, CNN-multichannel) on the Stanford sentiment treebank. Le and Mikolov [LM14] applied to the treebank logistic regression on top of their Paragraph Vector (PV) for distributed word representation. Finally, Multiplicative Recurrent Neural Network (DRNN)

[IC14], Constituency Tree-LSTM (CT-LSTM) [TSM15], and Dynamic Memory Network (DMN) [KIO+16] have also been applied to the Stanford sentiment treebank.

Model	Binary	Fine-grained
NB	81.80%	41.00%
BiNB	83.10%	41.90%
SVM	79.40%	40.70%
RecNTN	85.40%	45.70%
Max-TDNN	77.10%	37.40%
NBoW	80.50%	42.40%
DCNN	86.80%	48.50%
RNN	82.40%	43.20%
MV-RNN	82.90%	44.40%
SCNN	82.00%	43.50%
CharSCNN	82.30%	43.50%
SCNN ph.	85.50%	48.30%
CharSCNN ph.	85.70%	48.30%
CNN-rand	82.70%	45.00%
CNN-static	86.80%	45.50%
CNN-non-static	87.20%	48.00%
CNN-multichannel	88.10%	47.40%
PV	87.80%	48.70%
DRNN	86.60%	49.80%
CT-LSTM	88.00%	51.00%
DMN	**88.60%**	**52.10%**
GRU_{GloVe}	84.13%	45.89%
DNC	85.22%	46.78%

Table 8.8: Accuracy achieved by the compared methods on the Stanford sentiment treebank. Binary and fine-grained refer to 2-class and 5-class in-domain sentiment classifications respectively. The best accuracy is in bold.

Table 8.8 shows the comparison between the MemDNN architectures and the mentioned methods. Gated recurrent unit and differentiable neural computer achieve comparable performance in both binary and fine-grained configurations. The accuracy of differentiable neural computer is just about 1% higher than the accuracy of gated recurrent unit. They perform similarly to most of the other techniques, but are not definitely the best methods for single-sentence sentiment classification. This is probably due to the absence of a specific mechanism to take sentence syntax into account, and to the small amount of training data, which is an obstacle to the performance of both gated recurrent unit and differentiable neural computer. Just look at the in-domain experiment on the Amazon reviews (Figure 8.11), where they have been outperformed by Markov chain with small-scale data. One might argue that the treebank has

8,544 instances, but we should not forget that they are single-sentences, not whole and usually longer reviews (i.e. documents) as in the Amazon corpus. The best algorithm turns out to be dynamic memory network, which performs better than the other techniques in both binary and fine-grained configurations. This is not surprising, since dynamic memory network includes a memory mechanism to store and preserve relevant information through time and has also been proved to work well with single-sentences [KIO+16].

8.4.7 Final remarks

In this section, an investigation on recent deep learning advances has been carried out, emphasizing the characteristics that mostly affect the identification of sentiment. In particular, memory-based deep neural networks have been combined with techniques to learn distributed text representations, in order to exploit the advantages of both. To enhance the cross-domain effectiveness of sentiment classifiers built upon memory-based deep neural networks, fine-tuning has been performed on a small set of labeled target instances. Fine-tuning, GloVe word representation, and the ability of MemDNNs to capture relevant sequential information, aid the inter-domain alignment and bring to outstanding cross-domain document classification results.

The memory-based classifiers have also been employed on very large data sets (e.g. million instances), assessing their document-level performance in an in-domain configuration. Binary and fine-grained experiments have been carried out. The outcome has been compared with several variants of character-level convolutional neural networks. The classifier based on differentiable neural computer outperforms the state of the art with reference to both binary and fine-grained configurations, whereas gated recurrent unit with GloVe feature weights initialization achieves comparable performance with most techniques.

The experimented MemDNN methods can be applied to any text, whatever its length and structure. For this reason, single-sentence sentiment classification has also been performed, using the Stanford sentiment treebank as a benchmark dataset. The accuracy achieved by memory-based deep neural networks is comparable with the state-of-the-art methods in both binary and fine-grained settings.

Part III

Text Mining Approaches for Stock Market Analysis

9 Methods for DJIA index prediction

As discussed in Chapter 4, stock market prediction was historically accomplished either via trend analysis, that is, monitoring stock price variations through time, or via financial news analysis. More recently, researchers leveraged information extracted from social media, in particular Twitter, to design new forecast models for the stock market.

In this chapter, a novel approach is introduced that exploits data from Twitter to identify threads that significantly affect the Dow Jones Industrial Average (DJIA) index. Intuitively, the method is based on training an intermediate classifier on millions of tweets posted in the first seven months in 2008. Analyzing the results of this first classification, a pruning scheme is created that is based on four groups of tweets depending on the classification outcome, namely true and false positives, and true and false negatives. The training set is subsequently transformed by removing irrelevant tweets, considered as noise. This technique is applied both to individual tweets and to tweet aggregations considered as training instances, then used to setup a trading strategy.

9.1 Background

In order for readers to better understand the methods for DIJA index prediction we will introduce below, this section gives an overview of noise detection methods. On the other hand, some approaches that use Twitter for stock market analysis have already been discussed in Section 4.4.2, pointing out that Twitter can really be useful for prediction purposes.

9.1.1 Noise detection

Noise detection is a topic of interest since the dawn of information retrieval. There is a large literature regarding the detection of noise in data mining and especially in data clustering, although they sometimes refer to slightly different tasks, for instance outlier detection, novelty detection, anomaly detection and so on. Some methods have also been applied to text mining, generally for the recognition of noisy features [SR11] or for novelty detection [MS03], i.e. the discovery of unknown data that a machine learning system has not yet been trained with.

In a vector space model representation, noise removal can be addressed at two levels: at the feature level and at the instance level. With reference to the former, useless and non-informative words are removed: normally this problem is addressed with a lists of stopwords and feature selection schemes [Yan95, GM04]. With reference to the latter, non-informative documents are removed, which can potentially hinder the classification model. Noise detection techniques can ideally be used here without considering the textual nature of features. In literature, large amount of methods exist, for example K-nearest neighbors [KN98, RRS00], neural networks [BJC98, MNS05], decision trees [SR90, Joh95], support vector machines [TD99, SHX02] and bayesian networks approaches [BHMY99, DS07] can be used. In-depth descriptions of such techniques have been reported in some surveys [CBK09, MS03].

9.2 Detecting and mining relevant tweets

The spread of social networks and micro blogging enables people to share opinions and moods, creating very large and constantly updated textual corpora. Sentiment analysis techniques seek to extract emotional states or opinions expressed in each text document and create a collective social emotional state.

Can the trend of social emotional state predict the macroscopic evolution of global events such as some economic indicators? Recent studies have answered affirmatively to this question. In particular, Liu et al. [LHAY07] used a Probabilistic Latent Semantic Analysis (pLSA) model extracts sentiment indicators on blogs that predict future sales, Mishne et al. [MDR$^+$06] showed how the movie sales can be predicted through assessments of blog sentiments; similarly, Asur and Huberman [AH10] showed how public sentiment about movies expressed on Twitter can actually predict box office receipts. Gruhl et al. [GGK$^+$05] assessed the predictability of book sales using online chat activities. But all that glitters is not gold: Gayo-Avello [GA12] criticized some literature on this topic, showing results that are unpredictable, for instance the prediction of an election. Indeed, tweet analysis can help understanding the political popularity, but has not consistently been able to predict the results so far.

Some works have recently studied the correlation between sentiment extracted from Twitter and socio-cultural phenomena [BMP11], such as the popularity of brands [GSZ13], and the correlation between public mood on Twitter and Dow Jones Industrial Average (DIJA) [BMZ11]. Differently from previous works that predict the DIJA trend by computing people sentiment or mood from their opinions expressed through tweets, a simpler method is introduced here, based on mining techniques and text similarity measures for the characterization and detection of relevant tweets with respect to increments or decrements of DJIA. In particular, as far as the selection of tweets is concerned, our method includes a noise detection approach to filter out irrelevant tweets for DJIA prediction.

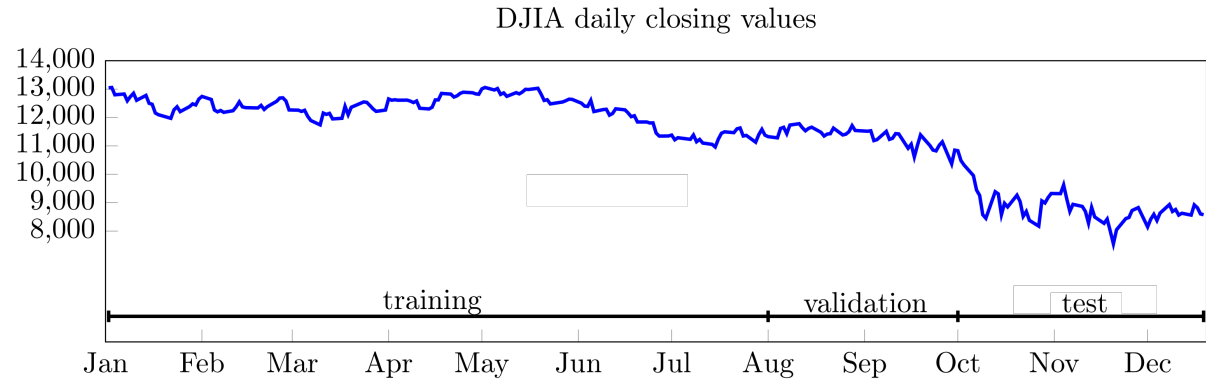


Figure 9.1: Daily closing values of DJIA prices, and splits of the benchmark dataset. *Source: [DMPP17a].*

9.2.1 Benchmark text set

The same set of tweets posted in 2008 and used by Bollen et al. [BMZ11] is employed. It includes ten million tweets posted by approximately 2.7M users from January 1^{th} to December 19^{th} 2008. Following the pre-processing applied by Bollen et al., only tweets in English that contain explicit statements of the author's mood state are taken into consideration, i.e. those that contains one of this expressions "i feel", "i am feeling", "i'm feeling", "i dont feel", "I'm", "Im", "I am", and "makes me". Tweets containing links or addressing the content to other users are removed. Tweets are tokenized in single words, and emoticons are also taken into account (Table 9.1), using three different tokens as done by Oliveira et al. [OCA13].

Token	Emoticons
emotHappy	:) :-) :D :-D (: (-:
emotSad	:(:-(:'(:'-( ):)-:)':)-': D: D-:
emotHeart	< 3

Table 9.1: Emoticons treated and related tokens.

Figure 9.1 shows the daily closing values of DJIA index. In comparison with the work by Bollen et al., a much smaller training set is considered in order to rely on a wider test set to properly evaluate the models' prediction capability. The benchmark dataset has been split into i) a training set including the first seven months of the year (from January 2^{nd} to July 31^{st}) to create the prediction models; ii) a validation set with two months, August and September, to tune the models and apply noise detection; iii) a test set with the latest three months, from October 1^{st} to December 19^{th}, larger than in [BMZ11], where only 19 days in December (15 of opening stock market) have been used.

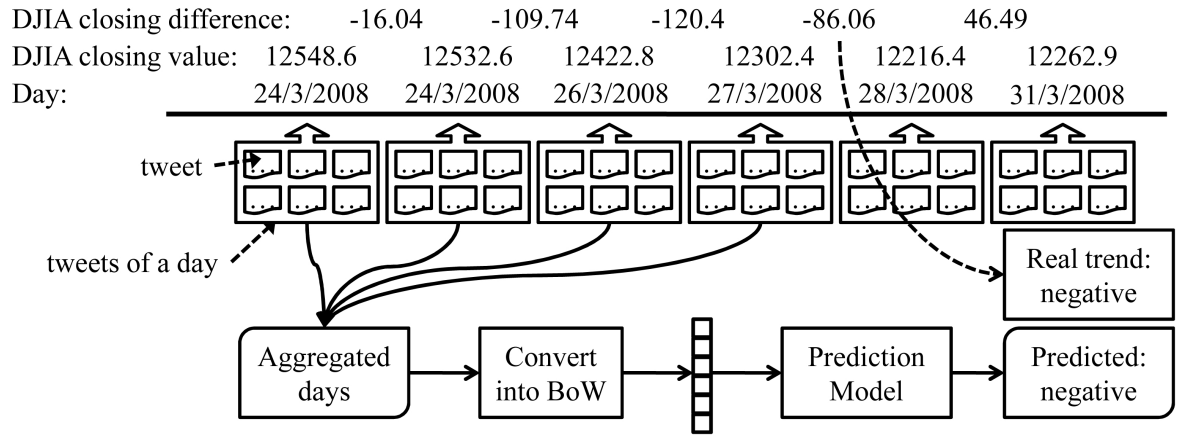


Figure 9.2: Diagram of the DJIA prediction process through tweets aggregation. In this example, the system predicts the DJIA trend for 2008/03/28 using the aggregated tweets posted in the previous four days (i.e. $l = 0$ and $a = 3$). *Source: [DMPP17a].*

9.2.2 Vector space model construction

Tweets are grouped according to the publication date and provide the information base to generate future predictions on the stock market. As shown in the experiments by Bollen et al [BMZ11], the highest correlation between social mood and DJIA is obtained by grouping tweets of several days and shifting the prediction for a certain time lag. Thus it becomes interesting to evaluate the accuracy of predictions considering the following parameters in the forecasting model:

- Lag (l): temporal translation from the forecast date, $l = 0$ means the day before prediction.
- Aggregation (a): number of days to be aggregated in order to make a prediction, $a = 0$ means only one day.

As a simple example, assume that we consider $l = 1$ and $a = 2$. To make a prediction on day t, tweets published in the days $t-2$, $t-3$ and $t-4$ will be considered. The range of days considered for the prediction of day t will be: $[t-1-l-a, t-1-l]$.

According to the two previous parameters, the tweets related to the prediction of a given day (in the previous example, the tweets in $t-2$, $t-3$ and $t-4$) are collected in a single bag-of-words. Given the high number of tweets available, a dimensionality reduction technique is required. Once the tweets have been selected, stopwords are removed, stemming is performed, terms are weighted using tf-idf, and only n_f of them (i.e. those with the highest weights) are selected.

At the end of each day d, from the mentioned aggregation of tweets, the method predicts the sign of the difference between the unknown DIJA closing price at day $d+1$ and the known closing price at day d. The proposed DJIA prediction process is summarized in Figure 9.2.

9.2.3 Detecting noisy tweets

Twitter provides a great deal of information, but it is necessary to understand what is useful for a given analysis and what is not. Therefore, a noise detection method is proposed here to define which tweets are useful for DJIA prediction. The idea can be summarized in few steps:

1. Once the data representation has been created, as described in Section 9.2.2, a classification model is trained and applied on the test set.

2. Four prototypes can be created, one for each possible classification outcome, namely true positive (TP), true negative (TN), false positive (FP) and false negative (FN). Each prototype is a bag-of-words that includes all the test instances, i.e. tweets of the a days before each prediction.

3. The prototypes are used to discover the noisy tweets within the dataset. Such a method is applied at two different levels: i) at the tweet level, and ii) at the instance level. The former consists in removing from the dataset those tweets with cosine similarity less than a threshold τ_g with respect to the good prototypes (TP and TN), or greater then a threshold τ_b with respect to the bad prototypes (FP and FN). The latter consists in removing from the training set instances similar to the bad prototypes.

4. A new prediction model is trained on the cleaned dataset, using the training and validation set, then it is used for the classification of test instances.

9.2.4 Experimental evaluation

The effectiveness of the prediction is tested by varying i) the classification algorithm: decision trees and support vector machines have been used, relying on Weka [FHH+05] implementations (i.e. J48 and SMO respectively); ii) the number n_f of features (i.e. words) selected in the dataset; iii) the aggregation a; iv) the lag l parameters on the data cited above.

Before the application of the noise detection method, a simple prediction model is tested, based on the vector space model described in Section 9.2.2. Table 9.2 and Table 9.3 show the best results obtained by the two supervised algorithms varying the respective parameters. A first noteworthy aspect is the aggregation parameter, that gives best results with three days gathered. This outcome confirms the analysis by Bollen at al. [BMZ11], meaning that there is strong correlation between the information extracted in the last few days and the outcome of a market trading day. In other words, the stock market seems to be affected by the information, events, moods of the previous days. Also, the best accuracy obtained by the decision tree model is notable: with just 500 features, it has achieved f-Measure of about 80%. From now on, tests will be performed using the best combination of parameters shown in Table 9.2 and Table 9.3.

Once the best model has been identified, the noise detection method is applied in order to clean the dataset. The idea is to analyze the predictions made on the test set in order to define

Aggr	Lag	n_f feat	fMeasure
3	0	500	0.799
3	1	2000	0.736
3	0	1000	0.700
0	2	500	0.668
0	2	2000	0.660
2	2	500	0.657
3	2	2000	0.653

Table 9.2: Tuning a decision tree.

Aggr	Lag	n_f feat	fMeasure
2	1	1000	0.682
1	2	2000	0.668
3	2	1000	0.649
2	1	2000	0.649
1	3	2000	0.643
0	2	2000	0.642
2	1	500	0.642

Table 9.3: Tuning a support vector machine.

four groups of predictions, and use them to find useful tweets, or tweet aggregations, in the dataset.

First, the test set is divided based on the prediction outcome. Among the tested instances, only the predictions where the confidence of the classifier is at least 90% have been selected, so as to retrieve only the surest among them. The selected instances are then grouped based on the outcome (i.e. TP, TN, FP, FN). In order to assess the assumptions and the quality of the groupings made, cosine similarity has been calculated both between instances of the same group and between belonging to different groups. The instances belonging to the same group were supposed to have high similarity, whereas dissimilarities were supposed to arise by comparing instances of different groups. The comparisons are shown in Table 9.4; the main diagonal contains those between instances belonging to the same group. It should be noted that such similarities are significantly greater than in the other comparison. This outcome supports the hypotheses underlying the noise detection method introduced in Section 9.2.3.

The first noise detection experiment has been made comparing the single tweets within the dataset (both training and validation sets) with the four prototypes created aggregating the instances of the four groups of predictions analyzed above. A double experiment has been conducted: i) keeping only the tweets similar to the two good prototypes, i.e. the tweets whose cosine similarity with respect to TP or TN overcomes a threshold τ_g; ii) discarding all the tweets similar to the bad prototypes, i.e. the tweets whose cosine similarity with respect to FP or FN overcomes a threshold τ_b. Figure 9.3 shows the obtained results in both experiments, varying the thresholds. Unfortunately, the results do not show an improving trend by using this noise

	TP	TN	FP	FN
TP	0.819	0.828	0.779	0.772
TP	0.823	0.914	0.776	0.738
FP	0.779	0.776	0.848	0.770
FN	0.772	0.738	0.77	0.912

Table 9.4: Comparison between instances (i.e. aggregated tweets) belonging to the different four groups (i.e. true positives, true negatives, false positives, false negatives). Each cell represents the average cosine similarity computed over all the respective couples of instances.

detection technique.

A further proposal to detect and remove noise is based on the idea that some training instances, such as outliers or noisy tweets, could compromise the accuracy of the prediction model. In this experiment, the instances similar to the bad prototypes that could negatively affect the model have been removed from the training set of the final classification model. Figure 9.4 shows the results obtained with the best tuning using both a decision tree and a support vector machine, varying the threshold τ of the noise detection algorithm. Results show a noteworthy improvement using the noise detection method. In particular, using the decision tree, we achieve $fMeasure = 0.889$, which corresponds to a 10% improvement with respect to the results obtained in tests without the training set cleaning techniques. Similar considerations can be done when using a support vector machine. In this case, the improvement is even greater, since we started from $fMeasure = 0.682$ and, with a 27% improvement, we obtain a maximum of $fMeasure = 0.867$ when using the noise detection algorithm. Analyzing the results obtained by the best model, we found that the $fMeasure$ related to the prediction of a positive market day is 0.848, whereas that related to the prediction of a negative day is 0.912. The precision of predictions on the validation set is 88.9%, that is higher than the precision obtained by Bollen at al. in [BMZ11], i.e. 86.7%.

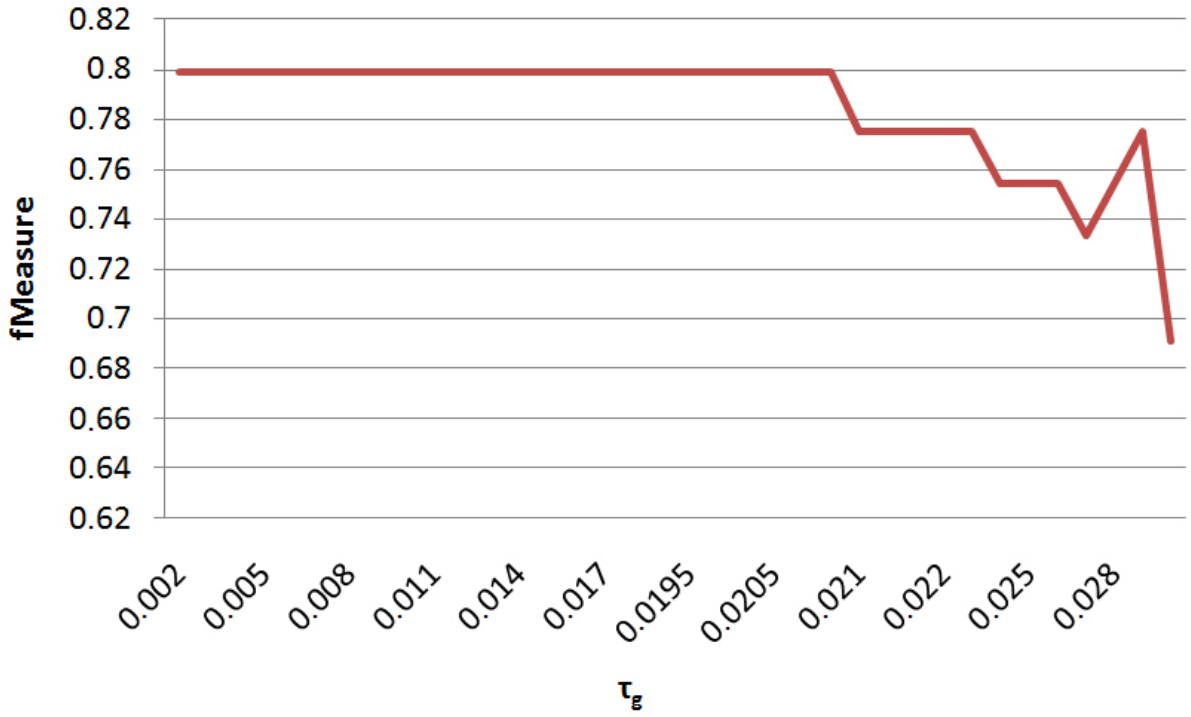
A fair comparison with the work of Bollen et al. can actually be done considering the same test set they used, i.e. the 19 trading days in December 2008. Using such a test set and training our method with the first 11 months of the year, a perfect classification (100%) of the 19 trading days is achieved, showing a sharp improvement with respect to the 86.7% obtained by Bollen et al.

9.3 A trading protocol for DIJA

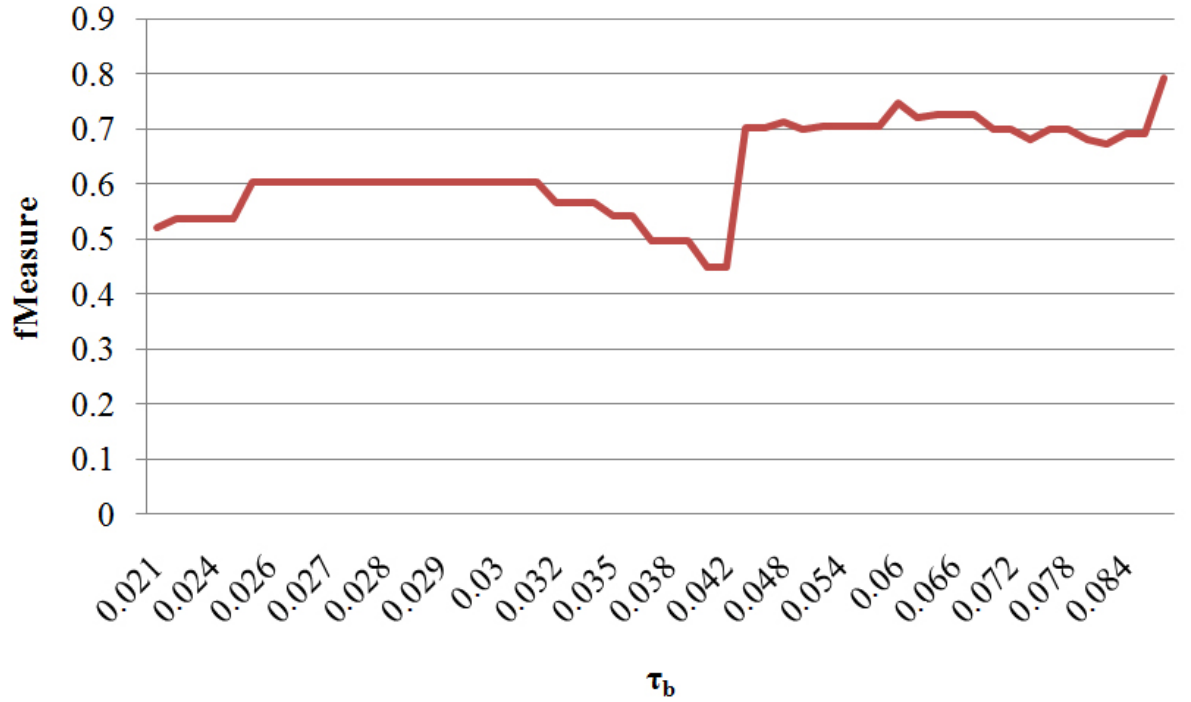
This section assesses the return on investment (ROI) achieved by trading Dow Jones Industrial Average (DIJA) according to the Twitter-based method introduced in Section 9.2.

9.3.1 Protocol

The accuracy of a predictive regression model can be measured in different ways, such as computing RMSE (Root Mean Square Error), evaluating the error in Dow Jones points, or using

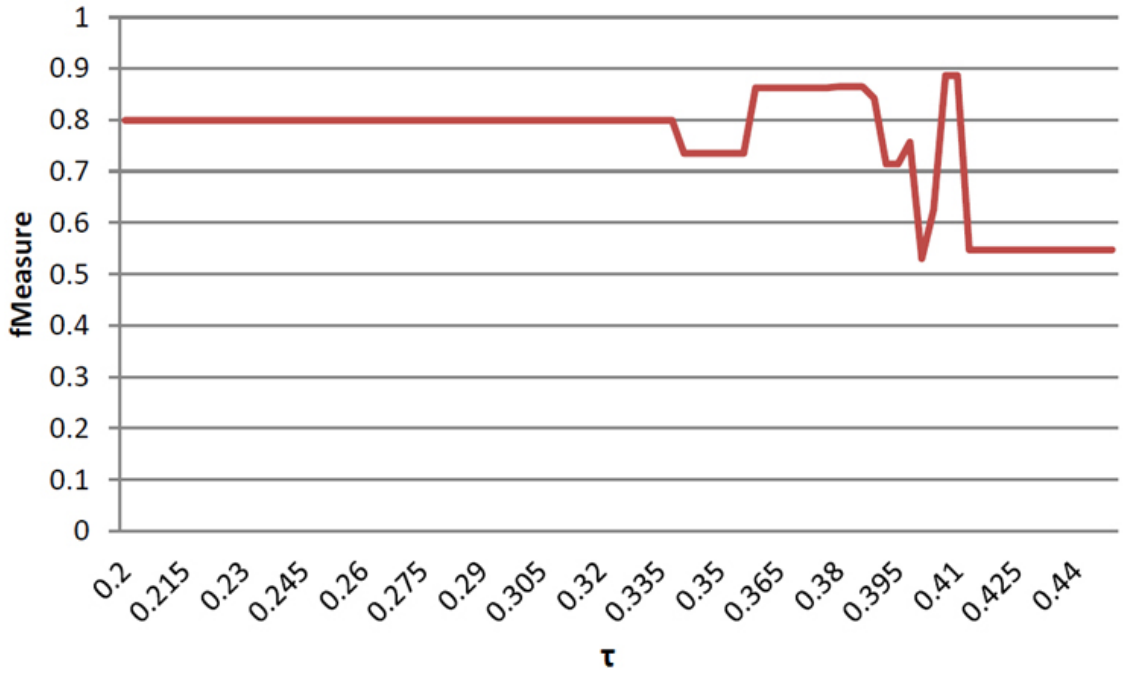


(a) Mantaining tweets similar to the good prototypes.

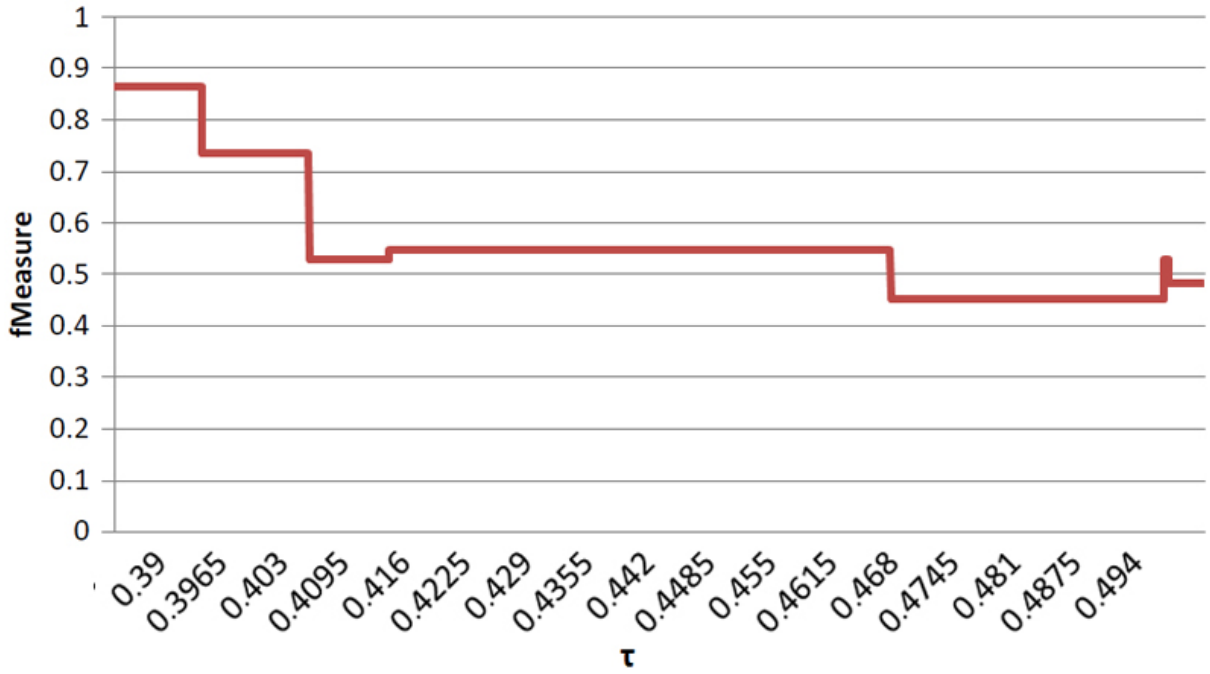


(b) Discarding tweets similar to the bad prototypes.

Figure 9.3: Noise detection experiments at the tweet level. *Source: [DMPP17a].*



(a) Decision tree supervised model.



(b) SVM supervised model.

Figure 9.4: Noise detection experiments at the instance level. *Source: [DMPP17a].*

the sign of the difference among consecutive closing prices. As pointed out in Section 9.2.2, we chose the last option. Indeed, what is ultimately important to measure the effectiveness of a prediction model is its capability of correctly driving trading actions to buy/sell shares. If the market is predicted to rise, the trading protocol receives a signal to buy from the prediction model, whereas a sell signal occurs when a drop is predicted. Sell/buy actions are performed without predefined rise/drop thresholds.

In our simple trading protocol, each buy/sell trade is conducted at the market opening value of each day d, and the forecast of DJIA movements consists in predicting the sign of the difference between the unknown closing price of day d, $closing^{(d)}_{unknown}$, and the known closing price of the previous day, $closing^{(d-1)}_{known}$. The action is taken as follows:

$$action\_at\_opening^{(d)} = \begin{cases} buy & \text{if } closing^{(d)}_{unknown} - closing^{(d-1)}_{known} > 0 \\ sell & \text{otherwise} \end{cases} \tag{9.1}$$

The trading protocol performs at most one action per day according to Equation (9.1). In each buy action, the whole capital is spent, including the possibly achieved gains (i.e. all in). At the first buy signal from the prediction model, the entire initial capital is invested, and any new subsequent buy signal is treated as a hold command. When a sell signal is received, the investment held is sold (i.e. all out), and subsequent sell signals are ignored until another buy is emitted. The protocol iterates in this manner until a stop signal occurs. To stop the protocol, it is sufficient to perform sell actions independently of the sign predicted by the model.

Gain/loss performance is measured by means of the standard Return On Investment (ROI) [OM06], which is defined as

$$ROI = \frac{Revenue\ after\ Investment - Cost\ of\ Investment}{Cost\ of\ Investment} \tag{9.2}$$

where $Cost\ of\ Investment$ is the amount of initial capital before the first trading action, and $Revenue\ after\ Investment$ is the amount achieved after the sale. For instance, a cost of investment equal to 1 at the opening value of day d, means that we have bought shares spending 1 (no matter the currency). If we sell the shares for 1.1 when the first sell signal occurs, ROI becomes $\frac{1.1-1}{1} = 0.1$, which means that we got a 10% profit. The sum of each ROI over the trading period is the percentage gain on the investment.

9.3.2 Experiments

Experiments have been performed comparing three prediction models: a random predictor of the DJIA rise/drop, and the two prediction models described in Section 9.2. Readers should recall that the model without the removal of noisy tweets achieved f-Measure equal to 79.9%, whereas discarding noisy tweets f-Measure reached 88.9%. We have performed the trading actions according to the protocol described in Section 9.3.1. The trading period coincides with the test interval of the prediction, i.e. from October to December 2008.

The performance of the three experiments has been measured according to the return on investment (ROI) described in Section 9.3.1. Table 9.5 reports the total ROI achieved by the above mentioned prediction models over the three months used as the testing period.

Prediction Model Type	ROI	Std.Dev.
Random	-0.097	0.138
No noise removal	0.562	
Noise removal	0.840	

Table 9.5: Total return on investment (ROI) in the three months used as the testing period. Three prediction models have been used: random trading actions, trading without removing noisy tweets, and trading with noise removal.

The random prediction model, which conducts completely random buy/sell trading actions, has been executed $10,000$ times, changing the random generator seed. On average, it achieves a 10% loss with a 14% standard deviation.

Our first prediction model, trained without the removal of noisy tweets, gains 56%, whereas the second one, trained removing noisy tweets from the training set as described in Section 9.2.3, gains 84%. Standard deviation has not been computed for the two latter results, as there are not random choices and experiments do not need to be repeated. As far as we know, both results outperform the state-of-the-art in trading with DJIA [OM06], even with respect to more recent advanced approaches based on deep learning and recurrent neural networks [FM18, BYR17]. In order to get an annual ROI, the above mentioned gains should be multiplied by four as they have been accumulated over a period of just three months.

Figure 9.5 reports the ROI time series produced by the three models over the test period, i.e. the last three months in 2008. The ROI time series of the random prediction model is always negative, and some days the loss exceeds 10%. The prediction model without noise removal initially has a slight loss, then it almost constantly increases. The prediction model with noise removal does not produce gains in the first week, then it grows really faster with respect to the two previous models.

9.4 Final remarks

In this chapter, we have developed a text mining method for the prediction of DJIA trend based on knowledge extracted from ten million tweets emitted within a year. The method recognizes and filters out irrelevant tweets that represent noise and would negatively affect the prediction accuracy. This is obtained by means of a noise detection technique, which acts both at the tweet level and at the instance level (i.e. aggregation of tweets).

Comparing the method with the same tweets dataset and DJIA trends used in [BMZ11], our simple classification model based on the vector space model has achieved about 80% accuracy. Applying the noise detection method, irrelevant tweets are filtered out from the training set,

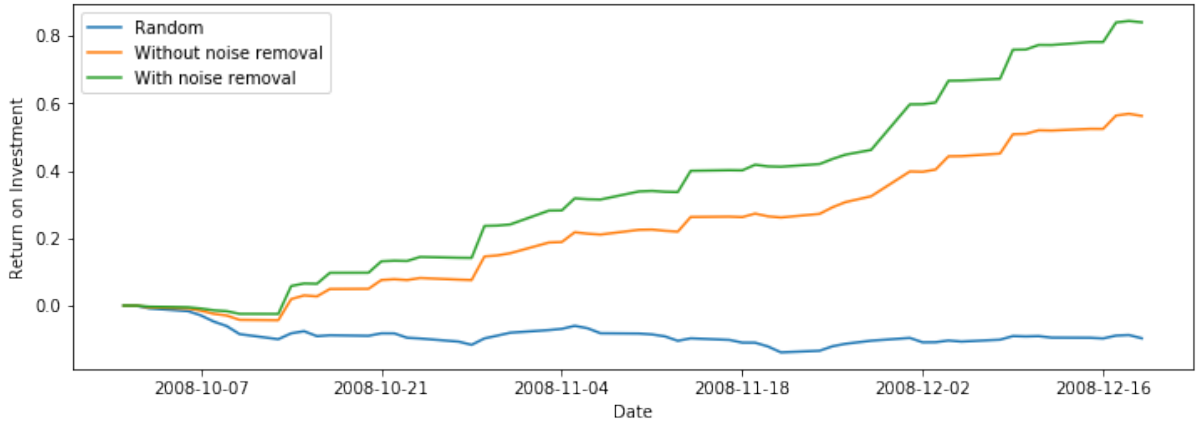


Figure 9.5: Time series of the return on investment (ROI) of three prediction models: random, without noise removal, with noise removal. The test period spans three months, i.e. from October to December 2008. *Source: [MPD$^+$].*

and accuracy grows up to 88.9%, outperforming both our base classifier and the best prediction method based on social network posts proposed in [BMZ11].

Finally, a trading protocol has been added in order to perform buy/sell operations on the basis of the prediction methods proposed. The trading experiments show the total return on investment (ROI) from the initial capital. The values without and with noise removal are 56% and 84% respectively, considering three months as the test period.

Part IV

Big Data Mining and Machine Learning Methods for Job Search

10 A skillset-based job recommender

In this chapter, a job recommendation approach is introduced, where the best job is suggested to a given candidate based on the semantic similarity between skills extracted from LinkedIn users. A job clustering is first proposed to find and group semantically similar jobs. Based on this clustering, a novel method is introduced to match candidates with jobs. The basic idea is to discover similarity between different job positions, and then to find out their latent associations with people's skills.

As far as we know, the most similar work with respect to our proposal and used data is by Bastian et al. [BHV+14], who constructed a folksonomy of skills and implemented a skill recommender system. In particular, their goal was to analyze the users' skillsets extracted from LinkedIn with the aim of helping users into profile skill filling. The authors pointed out the feasibility as well as the practical utility of job recommendation based on LinkedIn users' skills, without further discussion and detailed analysis.

10.1 Building a hierarchy of job positions

This section explains how LinkedIn public profiles can be used to build a hierarchical clustering of jobs. Hierarchical clustering aids to build a folksonomy (i.e. a user-defined taxonomy) of jobs. Such a folksonomy is useful to correlate distinct job positions, whereas only each person's job history is normally available. Therefore, a good job clustering is an important first step for systems that match candidates with jobs, namely job recommendation systems as well as recruitment systems.

10.1.1 Methodology

We describe here the process used to perform a clustering of job positions. We consider a set $\mathscr{U} = \{u_1, u_2, \ldots\}$ of user profiles (hence just profiles), where each of them can be considered as the description of a specific person.

To each profile u is associated a set $S(u)$ of skills, representing the competencies that a specific person claims to have. The same skill may be associated to more than one profile. The set containing all the distinct skills is denoted by $\mathscr{S} = \{s_1, s_2, \ldots\}$.

To each profile u is also associated the current job position of user u, denoted by $p(u)$. Since different users can perform the same job at the same time, many profiles can be associated with the same current job position. We denote by $\mathscr{P} = \{p_1, p_2, \ldots\}$ the set containing all the distinct job positions.

We are interested in obtaining a folksonomy of the possible job positions (hence just positions) from the available data. For this purpose, we perform a hierarchical clustering of positions in $\mathscr{P}$.

A structured representation of the possible positions is needed in order to measure their mutual distances. We extract a vector-based representation, where skills are used as features. Specifically, from the set $\mathscr{U}$ of known profiles, we build a $|\mathscr{S}| \times |\mathscr{P}|$ matrix $\mathbf{C}$ counting the co-occurrences between skills and positions within them. Values in $\mathbf{C}$ are computed as follows:

$$c_{i,j} = \left|u \in \mathscr{U} : s_i \in S(u) \wedge p_j = p(u)\right| \tag{10.1}$$

In practice, $c_{i,j}$ is the number of profiles having both s_i among skills and p_j as position. Each position is then represented as a weighted mix of different skills, according to those possessed by persons employed in that job.

Intuitively, skills within the set $\mathscr{S}$ can be semantically similar to each other or they can

even be synonyms: for example, "ms office" can be considered as strictly related to "ms word", while they are both quite unrelated to "psychology". Also, the same skill can be expressed with different names, e.g. "ms word" rather than just "word", or in different languages, e.g. "human resources" rather than "selezione del personale", which is the Italian translation of "human resources". The vector-based representation of positions can be improved by augmenting for each position (i.e. vector) the relevance of the skills (i.e. features) related to those explicitly included in the mix. This aspect has already been addressed in text analysis, where correlations usually exist between terms (i.e. features) appearing throughout text documents (i.e. vectors). A well-known technique to deal with such a problem is Latent Semantic Analysis (LSA), which employs Singular Value Decomposition (SVD) to obtain a lower-rank approximation of a term-document matrix [DFL$^+$88].

Equivalently, we apply LSA to the skill-position matrix $\mathbf{C}$ to obtain a transformed matrix $\mathbf{C}'$. We first decompose $\mathbf{C}$ into three factors.

$$\mathbf{C} = \mathbf{U} \cdot \Sigma \cdot \mathbf{V}^T \tag{10.2}$$

$\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices with eigenvectors of $\mathbf{C}$, while Σ is a diagonal matrix with eigenvalues. These matrices define a latent vector space, where skills and positions are represented by rows of $\mathbf{U}$ and $\mathbf{V}$, while values in Σ indicate the importance of each dimension in this space: lower values are supposed to represent dimensions mostly yielding noise instead of valid information. Setting all values of Σ to 0 except for the r highest ones, and multiplying back the three components, we obtain the transformed matrix $\mathbf{C}'$, which is a denoised approximation of $\mathbf{C}$ with rank r. For the r parameter, we choose the minimum value for which the sum of

retained eigenvalues is at least 50% of the total. The transformed matrix $\mathbf{C}'$, as the original one $\mathbf{C}$, contains for each position p_k a column vector $\mathbf{p}_k$ that represents the job itself.

The distance between two distinct job positions is computed as the inverse of their cosine similarity.

$$d(p_a, p_b) = 1 - \cos(\mathbf{p}_a, \mathbf{p}_b) = 1 - \frac{\mathbf{p}_a \cdot \mathbf{p}_b}{||\mathbf{p}_a|| \cdot ||\mathbf{p}_b||} \tag{10.3}$$

The mutual distances between positions are finally given as input to a complete-linkage agglomerative clustering algorithm, which extracts a dendrogram of the positions. Such a dendrogram has the form of a binary tree with positions as leaves.

10.1.2 Experimental setup

The methodology described in Section 10.1.1 to extract a hierarchy of job positions has been tested on a set of data extracted from LinkedIn. Operations have been carried out by custom software based on both Java and the open source R environment for statistical analysis.

The dataset used as a benchmark has been extracted from publicly accessible LinkedIn profiles of users from Italy: for each profile, the skillset declared by its owner, and the current job position have been considered. Skills and positions are specified by users as free text: some of them occur in many profiles, but most skills are only present in a few or just one profile, due e.g. to typos or uncommon names. The presence of different languages in the dataset constitutes another issue: many Italian users filled in the profile in their native language, whereas others used English to target a wider audience. Due to these aspects, the same actual skill or position can be found multiple times with different names.

We started from a preliminary dataset containing $51,000$ profiles as a training set and $37,000$ as a test set. The former initially included $2,410$ distinct positions: as some of them consisted only of punctuation (e.g. "?") or were not actual jobs (e.g. "mr."), we have removed them from the set $\mathscr{P}$ of admissible positions. Consequently, the training and test profiles having an inadmissible job as the current position have also been removed. In a second pre-processing step, we have removed the training profiles declaring less than three skills, in order to improve the quality of the knowledge base. This led to the removal of some positions and skills, which did not occur anymore in the training set. For consistency, we have also removed them from the test set, consequently deleting profiles having any of the removed positions. After such pre-processing operations, two disjoint groups of profiles suitable as the training and test sets are obtained. The former is used to compute similarities between skills and positions, and to build the hierarchy. The latter is used to evaluate the performance of the recommendation method that will be discussed in Section 10.2.

The final dataset is composed of $42,056$ training profiles and $30,639$ testing profiles; it contains $6,985$ unique skills, $2,241$ distinct positions, and at least 3 skills for each profile. The distributions of skills and positions are both highly skewed: for example, the most frequent position is "studente" (Italian translation of *student*) with $1,693$ training profiles and $1,383$ test ones, whereas other jobs have only one representative profile.

10.1.3 Job hierarchy

In order to infer a hierarchy of job positions from the training profiles, the method described in Section 10.1.1 has been applied. The goal is to obtain a consistent folksonomy where similar occupations are grouped and well separated from unrelated ones.

Due to the absence of a compatible gold standard, it is not possible to quantitatively evaluate the correctness of the inferred hierarchy. Therefore, we browsed through the obtained tree to check whether the obtained clusters are reasonable. We report in Figure 10.1, Figure 10.2 and Figure 10.3 some examples of the hierarchy, also showing the bottom-most binary splits between elements. Since job positions have both English and Italian names, a translation of the latter is provided in figures for readers' convenience.

Readers may note in Figure 10.1 and Figure 10.2 that the clustering algorithm mostly succeeds in outlining groups of related job positions. As the relatedness among jobs has been inferred by using their co-occurrences with skills within users' profiles rather than just lexically similar words, similar occupations are effectively grouped into the same cluster, although they are expressed with different terms. Analogously, the same positions with distinct English and Italian names are mostly successfully grouped. This outcome suggests that jobs can be frequently identified based on the skills of the users having such positions.

On the other hand, the clustering may also produce heterogeneous groups, including unrelated jobs, as can be seen in Figure 10.3. This outcome can be due to singularities in the co-occurrences between skills and positions within the training set. Job positions associated to an adequate number of profiles and skills have a consistent representation, whereas others that rarely occur in profiles are mostly associated with unrelated skills. For example, "personal trainer" has been considered similar to occupations dealing with computer networks. While it appears illogical, the cause can be found in the profiles used to infer the taxonomy. Two out of nine training profiles having "personal trainer" as the current position, declare IT and telecommunications-related skills such as "linux" and "tcp/ip". If no other occupation with sufficient occurrences is significantly similar to "personal trainer", this position will be grouped with unrelated jobs that share a significant part of their skillset with "personal trainer". Another example is "panettiere" (Italian translation of "baker"), which is found as the current employment in two training profiles only. While one of them explicitly includes "bakery" among the abilities, the other skills in both profiles are unrelated, mostly very generic, such as "teamwork" and "problem solving", which can equally be linked to other uncommon positions.

Summing up, the hierarchy successfully delineates groups of similar job positions, although few clusters of unrelated occupations may also be produced.

10.2 Job recommendation

In this section, a novel method for job recommendation is proposed, which exploits the clustering of job positions illustrated in Section 10.1 to recommend the most appriopriate jobs to a given candidate.

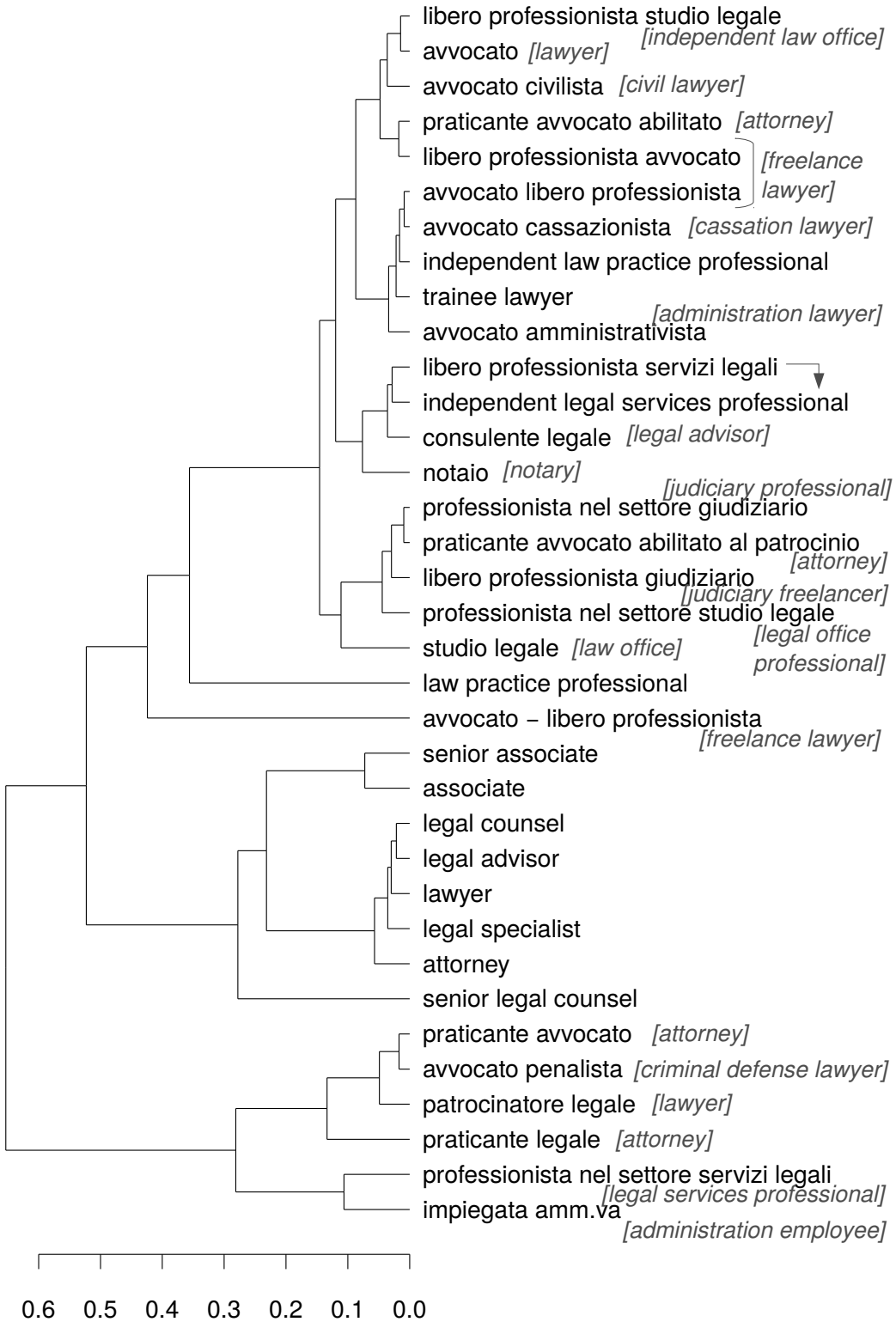


Figure 10.1: A part of the hierarchy containing legal jobs. *Source: [DMP+16].*

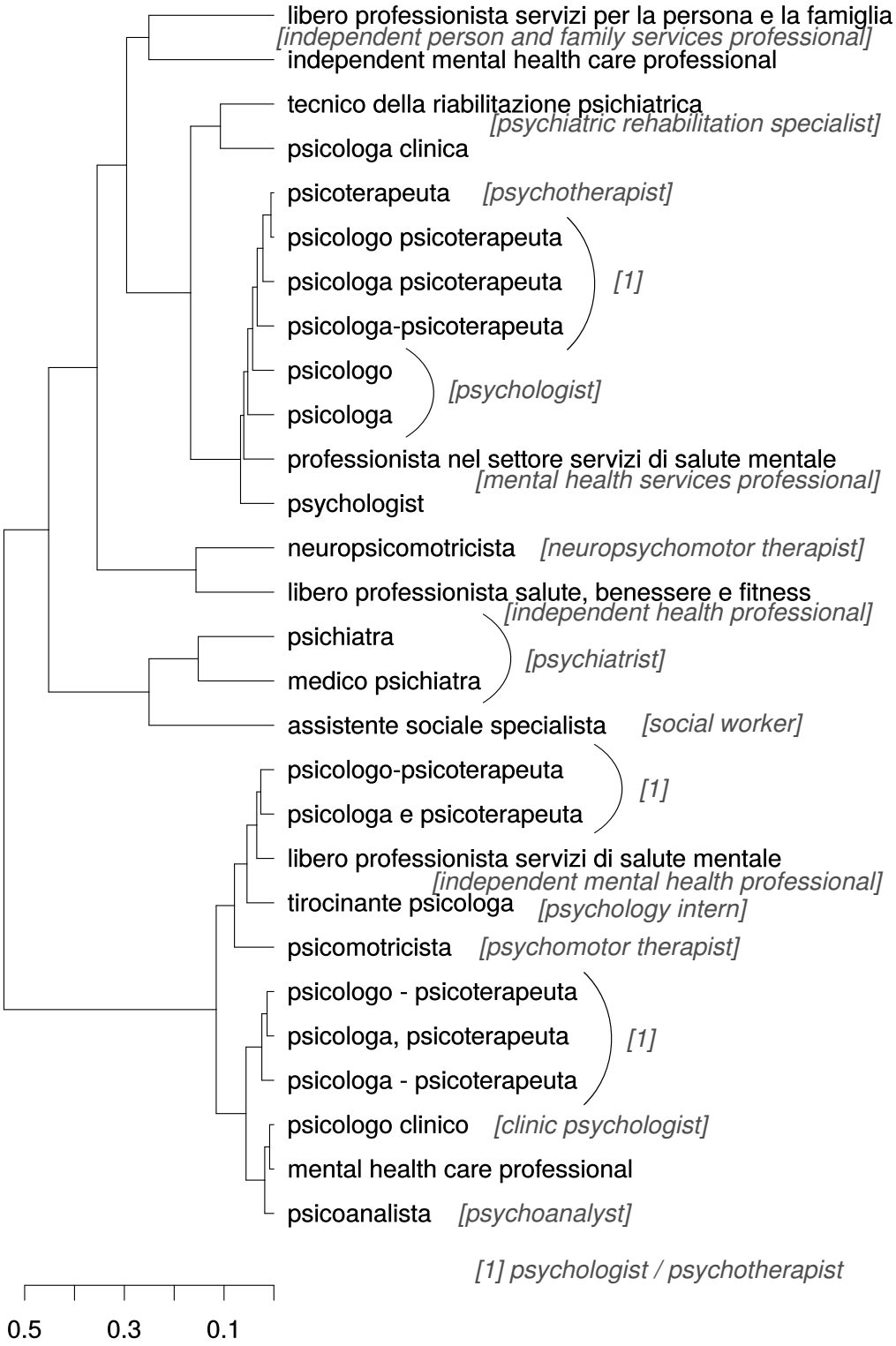


Figure 10.2: A part of the hierarchy containing jobs related to mental health care.

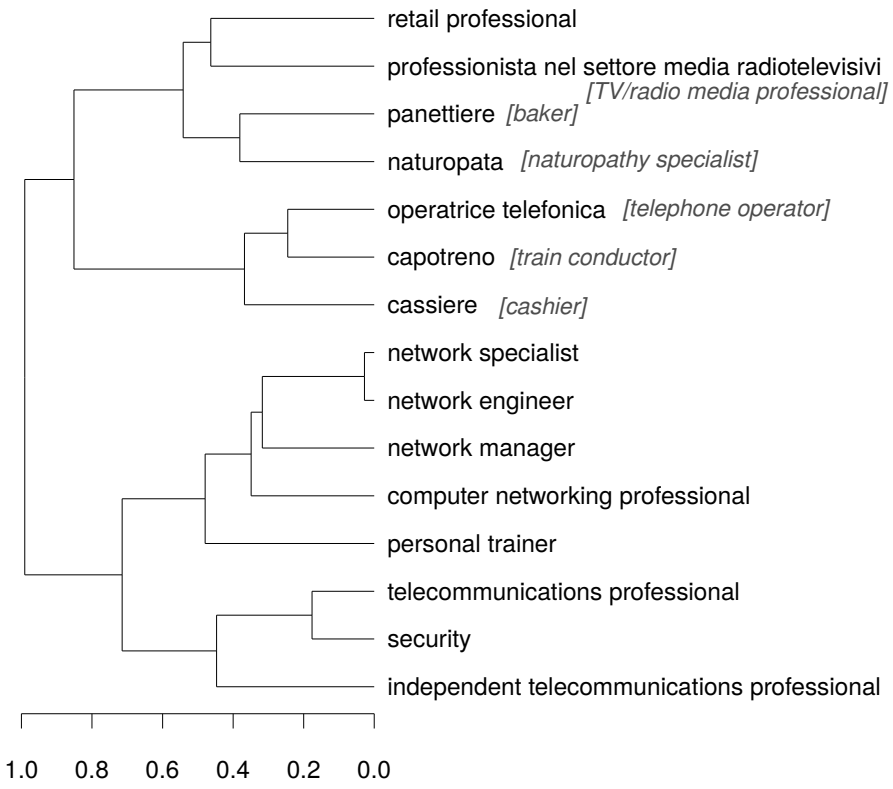


Figure 10.3: A part of the hierarchy containing heterogeneous jobs. *Source: [DMP+16].*

10.2.1 Methodology

Other than extracting a consistent hierarchy of positions, the knowledge of a set of profiles can be used to infer the most advisable job positions for any other profile u_e, whose skillset $S(u_e)$ is given. In other words, a job recommendation system can be produced, where the best jobs are suggested to users according to their skillsets. While the current positions in profiles are assumed to be correct (i.e. the best possible), it should be noted that multiple positions can be recommendable given a skillset. Therefore, a job recommender could possibly suggest a set containing the most likely job positions.

In our recommendation method, both positions and profiles are represented as comparable vectors, and the jobs with the most similar vectors are selected for each profile The skillset $\mathscr{S}$ is used as the feature set. For each profile, a ranking of job positions is output, whose only the first k items are usually considered.

Each profile u_e is simply represented by a binary vector $\mathbf{u}_e$, whose values are 1 in correspondence of skills known by the user, and 0 elsewhere.

Regarding the vectors of job positions, we reuse the results of position clustering: one option is to use the columns of the original skill-position co-occurrences matrix $\mathbf{C}$; another is to use its low-rank approximation $\mathbf{C}'$, computed by means of LSA as described in Section 10.1.1.

In both cases, we compare the user's skillset $\mathbf{u}_e$ with each column $\mathbf{p}_j$ by means of cosine similarity. For a given number k of positions to be recommended, we return the set $R_k(u_e)$ of k positions whose vectors are most similar to $\mathbf{u}_e$. Such positions represent the recommendation for u_e.

10.2.2 Results

Job recommendation is performed using the same pre-processing and partitions (Section 10.1.2) employed to build the hierarchy of job positions. Specifically, we compute job recommendations for profiles of the test set, hereby denoted by $\mathscr{U}_{\text{test}}$, comparing the output of our method with the known one.

For the experimental evaluation, we consider the current occupation of each user as the correct answer that should be given by the recommender, ignoring further information. Readers should consider that the identification of such a specific job position is very hard for the recommender. As discussed above, due to the use of free text, many synonyms and misspelled variants may be used to denote the same job, but they will actually be considered as distinct elements in $\mathscr{P}$. The recommender may also suggest a "wrong" job, different from the actual one, but requiring a very similar skillset. Although in such a case the recommendation is considered as wrong, it will probably be reasonable in a practical use case. Ultimately, persons can be practicing jobs that are unrelated to their skills. These aspects may introduce outliers and potential errors in both training and test data, which negatively affect accuracy and are detrimental for quantitative evaluations.

The algorithm outputs the k most appropriate job positions. The actual position of a test

profile could either be among the recommended or not, and we want to evaluate how frequently the recommender hits the actual positions of test profiles, varying the value of k. For k ranging from 1 to 50, we evaluated the recall@k, i.e. the ratio of test profiles w.r.t. their total for which the actual position is among the top k recommendations.

$$\mathrm{R@}k = \frac{|u \in \mathscr{U}_{\text{test}} : p(u) \in R_k(u)|}{|\mathscr{U}_{\text{test}}|} \tag{10.4}$$

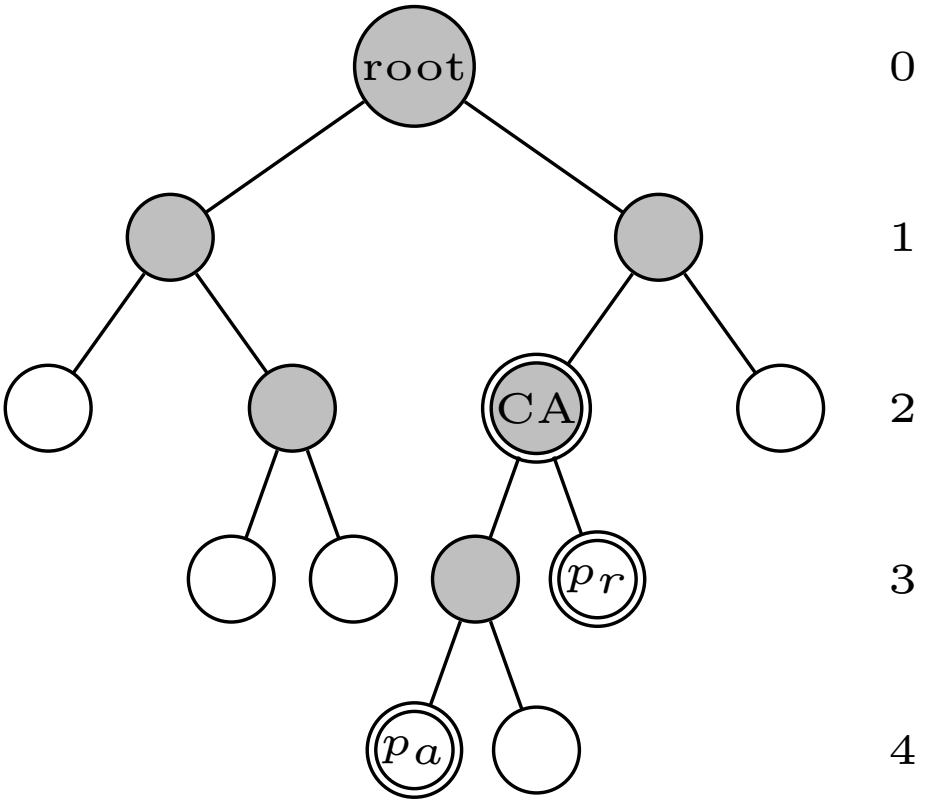


Figure 10.4: Illustration of hierarchcal recall: gray nodes in the tree are clusters, white nodes are single job positions (i.e. leaves), numbers on the right indicate the depth of the tree.

As discussed above, recommended positions can possibly be reasonable, in spite of being different from the one actually declared by the user's profile. Indeed, positions that are similar to the target one are usually good recommendations. We can leverage the previously computed hierarchical clustering of positions to evaluate how much a recommended position is close to the actual one. For this purpose, we use hierarchical recall (hR), proposed by Silla and Freitas [SF11]: given an actual position p_a and a single recommendation p_r, hR is the ratio between the depth (i.e. the distance from the root, denoted here by Δ) of their deepest common ancestor (CA) and that of p_a. For example, in the case depicted in Figure 10.4, hR is $2/4 = 0.5$, as the actual position is found at depth 4 and the deepest common ancestor between it and the recommended

position is at depth 2. In case of k recommendations for the same profile, the maximum hR is considered; for the whole test set of profiles, the mean is computed.

$$\text{hR@}k = \frac{1}{|\mathscr{U}_{\text{test}}|} \sum_{u \in \mathscr{U}_{\text{test}}} \max_{r \in R_k(u)} \frac{\Delta(\text{CA}(p(u), r))}{\Delta(p(u))} \tag{10.5}$$

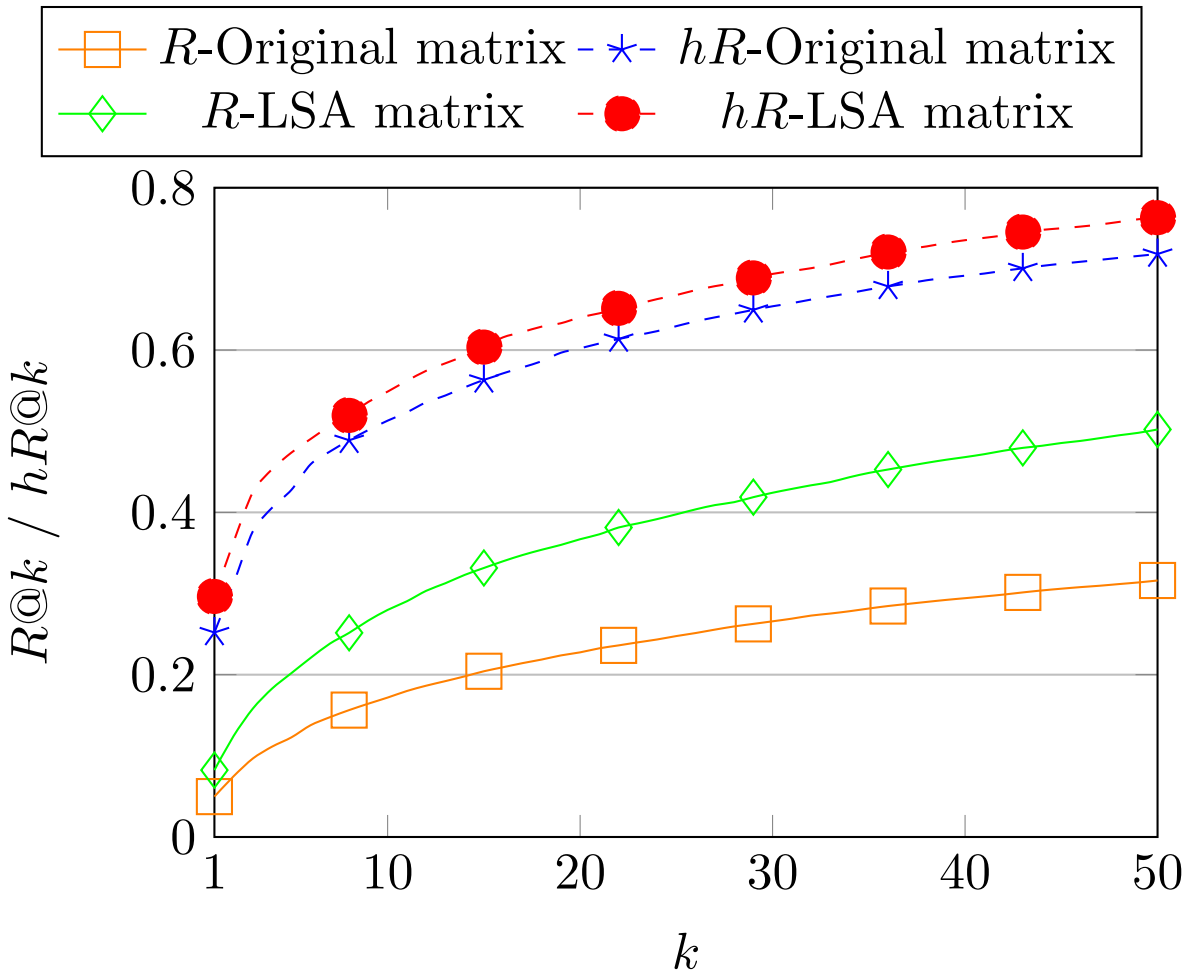


Figure 10.5: Trends of recall (solid lines) and hierarchical (dashed lines) recall for values of k from 1 to 50. *Source: [DMP+16].*

The recommendations of job positions for test profiles have been computed using both the described approaches, i.e. representing positions either with the original co-occurrences matrix $\mathbf{C}$ or with its low-rank approximation $\mathbf{C}'$ obtained by means of latent semantic analysis. In both cases, we compared recommendations with actual positions to compute traditional recall and hierarchical recall for values of k from 1 to 50. Table 10.1 reports the values that recall assumes for some specific k, whereas the plot in Figure 10.5 summarizes all the measurements.

	Original matrix		LSA matrix	
k	R@k	hR@k	R@k	hR@k
1	0.050	0.252	0.082	0.296
5	0.123	0.428	0.202	0.474
10	0.172	0.513	0.280	0.549
20	0.228	0.602	0.367	0.640
50	0.316	0.718	0.502	0.763

Table 10.1: Recall and hierarchical recall for some k values.

Skills (alphabetical order)	Known position	Top 5 recommendations
blogging, e-commerce, facebook, marketing communications, marketing strategy, social media, social media marketing, social networking	responsabile customer service (*customer service manager*)	(1) marketing manager (2) sales manager (3) titolare (*owner*) (4) export manager (5) agente di commercio (*sales agent*)
apache, arch linux, centos, cisco nexus, cisco switches, dns, f5 bigip, netbackup, network administration, radware, red hat linux, sophos, trend micro, vmware, vmware esx, vsphere	it system administrator	(1) system administrator (2) it manager (3) network administrator (4) system engineer (5) senior system engineer
adults, mental health, psychology, psychotherapy	psychologist	(1) psicologa (*psychologist*, woman) (2) psicoterapeuta (*psychotherapist*) (3) psicologa psicoterapeuta (woman) (4) psicologo (*psychologist*, man) (5) psicologo psicoterapeuta (man)

Table 10.2: Example of test profiles with skills, known positions and recommendations. English translations of Italian position names are reported in italic.

Comparing the results achieved with the two matrices, the usage of latent semantic analysis always appears to be beneficial for the recommendation. Indeed, it improves the representation of jobs according to their statistically estimated relatedness, and accuracy increases consequently. Therefore, the following analysis focuses on the LSA matrix.

Obviously, the accuracy grows as the number k of recommendations to be output increases, because it is more likely to hit the exact position or a very similar one. However, a smaller set of good recommendations can often be more valuable in practice than a larger one, which could more likely include improper elements. Looking at the traditional recall, readers can see that a single recommendation for each profile exactly matches the actual occupation in just 8.2% of the test cases. As the number of recommendations grows, the known position is more likely to be hit: this happens in about one case every five when $k = 5$, one every four when $k = 8$ and one every two when $k = 50$.

Comparing the traditional recall with hierarchical recall for equal values of k, the latter is

superior by a consistent gap, ranging between 21% and 27%. This outcome suggests that in many cases where the actual position is not recommended, at least one of the recommended jobs is very similar.

This phenomenon can also be observed by manually comparing recommendations with known positions. Table 10.2 shows, for some test profiles, both the actual known position and the recommended jobs. It should be noted that, although the method fails in getting the right occupation within the very top recommendations, the recommended jobs are anyway quite similar to it or even synonyms. In practical use cases, such recommended positions still are valid and reasonable for the given skillset.

10.3 Final remarks

In this chapter, we have introduced a novel job recommendation approach based on latent semantic analysis, evaluating its performance by means of a hierarchical job clustering, which is useful to correlate heterogeneous jobs. The basic idea of the method is to discover job similarities, then to find out their latent associations with users' skills.

For a matter of comparability, both people and jobs are represented as vectors of skills. The algorithm starts from a skill-position matrix built from training data and expanded by applying latent semantic analysis. Afterwards, cosine similarity between people and positions in the skill-position matrix is computed for each test instance. Thus, since the algorithm outputs how much jobs fit people's skills, an ordered list of the recommended jobs can be built according to their similarity with each person's skillset.

LinkedIn has been taken as the reference scenario to evaluate the method because of its widespread use, crawling real public profiles from which people's skills and current job positions can easily be inferred. Differently from other approaches [Chi99, MKWW06, PCG11], the proposed method does not require to manually collect data, since large-scale data both in terms of job positions and skills are already available on social networks. Finally, it should be noted that neither the hierarchical job clustering nor the job recommendation algorithm require parameters to be tuned. This makes the approach easy to use, and profitable in real and big data scenarios.

11 Modeling job transitions and recommending career pathways

Traditional job recommendation and recruitment systems focus on finding a match between candidates and jobs, based on a set of relevant characteristics, including users' requirements and competencies, job requirements, and possibly other constraints. This chapter provides a mechanism to match candidates with jobs that takes uncertainty about career choices into account.

A skillset-based approach is advanced to build a job graph from raw job postings, where job relationships are modeled in terms of skills. Then, the job graph, a candidate's past job history and career goal, and possibly other constraints, are given as input to a Career Pathway Recommendation (CPR) algorithm based on Markov Decision Process (MDP), which is able to recommend a personalized optimal policy for the candidate to reach her career goal.

The main steps of the approach, which will be carefully described below, can be seen in Figure 11.1. In order to preserve word order and to extract word semantics, a distributed representation of the words in job postings is learned without supervision. A clustering of job postings is performed to group similar jobs. A set of relevant terms (i.e. skills) is extracted for each cluster, and a skillset-based job graph is built accordingly.

When a candidate asks for a career pathway recommendation, her job history is exploited to build her profile in terms of skills. Then, the candidate's career goal is mapped to a state of the job graph, i.e. goal state. The candidate's profile, the goal state, and the job graph are used to create an MDP, which takes into account the possible choices that the candidate could perform to reach her career goal. An optimal policy can be extracted from the MDP and recommended to the candidate, along with the skills that the candidate will acquire during the intermediate career steps.

11.1 Modeling job transitions

This section introduces a mechanism to build a job graph from job-related information, so as to mine relationships between distinct jobs and to model the likelihood to move from one job to another. Such a mechanism relies on skills that are mined without supervision combining

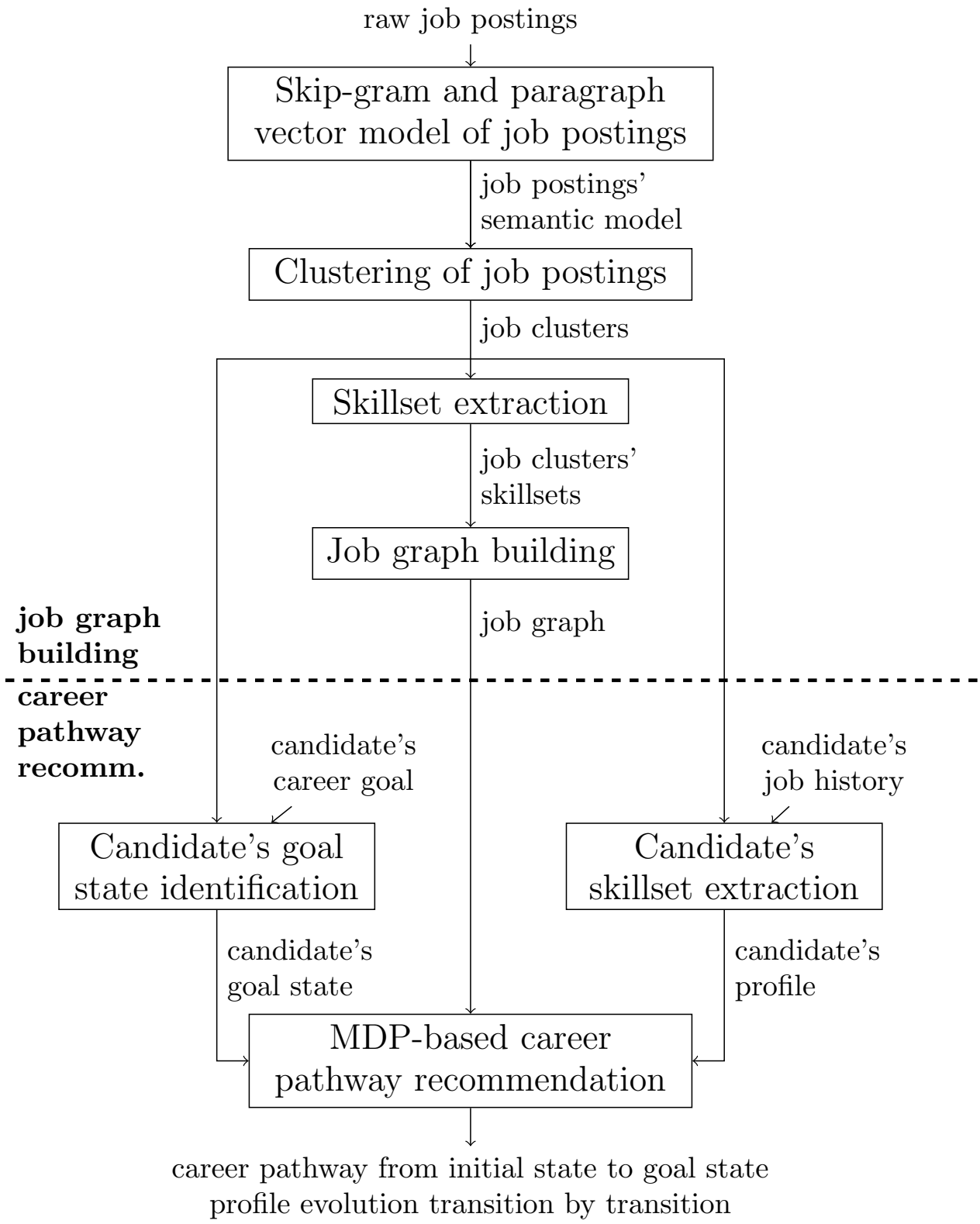


Figure 11.1: A schematic representation of the approach used to build a job graph and to exploit it for career pathway recommendation. *Source: [PAB+].*

the skip-gram model with a clustering algorithm, and then used to model job transitions. In particular, a clustering of job postings is used to mine similar jobs, which are encoded as word vectors and learned through the skip-gram model, so as to extract word semantics and discover phrases in text without supervision [MCCD13, MSC+13].

The most relevant terms of each cluster, which represent the required skillset by jobs in that cluster, are used to express the likelihood to move from one cluster to another, i.e. the likelihood that an application for a job in a cluster succeeds, given that the current job is in another cluster. Modeling job transitions paves the way for an emerging research thread, that is career pathway recommendation. A novel MDP-based career pathway recommender will be introduced in Section 11.2.

11.1.1 Clustering of similar jobs

The idea behind the construction of a job graph is to discover the characteristics and the relationships among jobs. On one hand, knowing the main features of a job allows candidates understanding whether they are ready to apply for it, along with the skills they need to strengthen in order to increase the likelihood to succeed in their application. On the other hand, identifying similar jobs is useful to provide candidates with many options to meet their profiles.

Let us assume to have a corpus of job postings in text format. Since the main goal is to extract job similarity, a semantic-bearing word representation is required. The advent of deep learning has brought very powerful techniques for text encoding, named distributed representations (aka word vectors), alternative to the bag-of-words model. Among those, local context window methods, such as the skip-gram model and the continuous bag-of-words model [MCCD13], have been proved to solve the main drawbacks of bag-of-words model, which loses the ordering of words and ignores their semantics. Word vectors have desirable algebraic properties, e.g. *king - man + woman* results in a vector very close to *queen* [MSC+13]. Moreover, the skip-gram model is able to discover phrases in text, e.g. *computer engineer* rather than just the words *computer* and *engineer*. These properties make the skip-gram model suitable to encode words in job postings. The whole job posting can finally be encoded as a paragraph vector, proposed by [LM14] to learn fixed-length representations of variable-length text.

Once a distributed text representation has been learned, the main features of job postings can be mined. Job postings usually include job-related information, typically unstructured or semi-structured, such as job title, description, requirements, location, salary, and so on. Let $\mathscr{J} = \{j_1, j_2, \ldots, j_n\}$ be the set of job postings, which are initially considered to be different. However, distinct job postings can refer to the same topic, e.g. *computer scientist* and *software engineer* refer both to the IT world. It may also happen that similar jobs are described using lexically different terms, e.g. *"SW engineer who is very good at programming"* and *"programmer with a strong software development background"* are very related from a semantic point of view, despite the fact that they do not share any word. A clustering of job postings can be performed to cope with such frequent situations, so as to form semantically similar groups. Formally, each job posting $j \in \mathscr{J}$ will be assigned to a cluster $c_i \in \mathscr{C}$, where $\mathscr{C} = \{c_1, c_2, \ldots, c_k\}, k \leq n$

is the set of clusters. After job postings have been clustered, the H most important terms $\mathscr{S}^{c_i} = \{s_1^{c_i}, s_2^{c_i}, \ldots, s_H^{c_i}\}$ can be retrieved from each cluster c_i. These terms represent the skillset that characterizes all jobs within the given cluster. In other words, they can be seen as the required skillset in order to get a job within the given cluster. Several techniques exist that aim at extracting relevant terms from unstructured data, including Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA), and Term frequency - Inverse document frequency (TfIdf). Such techniques can create job clusters $\mathscr{C}$ as well as the skillsets $\mathscr{S} = \{\mathscr{S}^{c_1}, \mathscr{S}^{c_2}, \ldots, \mathscr{S}^{c_k}\}$ that characterize them.

11.1.2 Building a job graph

From a theoretical point of view, a job graph could also be built straight away from single jobs $\mathscr{J}$, rather than from job clusters $\mathscr{C}$, i.e. $\mathscr{J} = \mathscr{C}$ when each job posting forms a standalone cluster. However, building the job graph from job clusters is beneficial, because semantically similar jobs are already grouped and denoted with the same skillset, and they can be considered as equivalent career steps. To clarify this aspect, let us consider *computer programmer* and *application developer* as examples of job positions. Although they are syntactically different, they are likely to refer to the same job, or at least to semantically similar jobs, whose skillsets are similar. Identifying their similarity is very important in terms of career recommendation. If they were considered as different, a *computer programmer* position could be recommended as the next career step to an *application developer*, and vice versa, although that recommendation will not be useful in terms of career progression. This will not happen if they are considered as the same job. Both the *computer programmer* and the *application developer* will end up with more useful recommendations, such as for instance *software analyst* or *project manager*. Moreover, considering *computer programmer* and *application developer* as the same job, if a recommender system suggests the former as the right job for a given candidate, she will automatically know that the latter is as good as the former.

After having pointed out the advantages of job clustering, let us describe how it can be used, together with the skillset characterizing each job cluster, to build a job graph. A job graph $\mathscr{G} = (\mathscr{C}, \mathscr{L})$ is a directed graph, whose nodes (aka states or vertices) represent job clusters $\mathscr{C}$ and whose transitions represent the likelihood to get a job in cluster $c_j \in \mathscr{C}$, starting from cluster $c_i \in \mathscr{C}$. The basic assumption here consists in modeling both vertices and transitions in terms of skills. As previously stated, each cluster c_i is associated with and characterized by a set of terms $\mathscr{S}^{c_i} = \{s_1^{c_i}, s_2^{c_i}, \ldots, s_H^{c_i}\}$. In other words, $\mathscr{S}^{c_i}$ can be considered as the required skillset to get a job $j \in c_i$. The state transition matrix is denoted by $\mathscr{L} = \{l_{ij}\}, \mathscr{L} \in [0,1]^{k \times k}$, where l_{ij} is the likelihood of getting a job in node c_j, given that the current job is in node c_i. This probability may depend on several factors, such as the different importance of skills, time and geographical constraints, just to mention some. Without loss of generality, here skills have been considered equally relevant, and other constraints have been ignored in order to keep the approach as simple as possible. Other ways to compute probabilities can easily be plugged in. The likelihood l_{ij} only depends on skillsets $\mathscr{S}^{c_i}$ and $\mathscr{S}^{c_j}$. In order to provide a more precise definition, let us

define the overlapping factor O_{ij} between a source node c_i and a destination node c_j:

$$O_{ij} = \frac{|\mathscr{S}^{c_i} \cap \mathscr{S}^{c_j}|}{|\mathscr{S}^{c_j}|} \tag{11.1}$$

Note that $O_{ij} \in [0,1]$, and, in general, $O_{ij} \neq O_{ji}$. In particular, $O_{ij} = 0$ when no skill in $\mathscr{S}^{c_i}$ is required by jobs in c_j, and $O_{ij} = 1$ when all required skills by jobs in c_j are already in $\mathscr{S}^{c_i}$. Trivially, $O_{ii} = 1$. The likelihood l_{ij} of getting a job in node c_j, given that the current job is in node c_i, is a function of the overlapping factor O_{ij}. Let us assume the probability exponentially increases with the overlapping factor, which is reasonable if the skills are equally relevant and other constraints are disregarded. Under this assumption, the likelihood l_{ij} can be expressed as follows:

$$l_{ij} = f(O_{ij}) = \frac{e^{O_{ij}} - 1}{e - 1} \tag{11.2}$$

Relying on Equation (11.2), the job graph is fully connected, because l_{ij} is computed for each $i, j = 1, 2, \ldots, k$. However, a threshold l_{thr} can be added to avoid less likely state transitions and to speed up computation, especially when a large number of clusters is considered and when very unlikely transitions exist.

11.2 MDP-based career pathway recommendation

In this section, a general framework for career pathway recommendation is introduced. Such a framework is based on a sound mathematical theory, namely Markov Decision Process (MDP). It provides a personalized recommendation to each user, given an user's profile, a career goal, possibly far off in the future, and the structure of the job graph described in Section 11.1.

MDPs are suitable for recommending a career pathway, because they are able to model uncertainty and rewards, both at the goal state and along the way. These capabilities are very important for a career pathway recommender. Rewards allow ranking career alternatives, identifying their goodness. However, it may happen that the job that apparently is the best option at some point of the pathway cannot be reached, or that such a job is not the global best option, in spite of being the local best. Modeling uncertainty allows dealing with such unexpected situations.

11.2.1 Recommendation requirements

The job graph described in Section 11.1 can be exploited by a pathway recommender to suggest an optimal policy for a user to reach her career goal. Careful readers may have noted that the job graph is unique, independent of users. In order to provide a user with a personalized recommendation, her past job history is required. Let us hypothesize the user u has performed m jobs in her history $\mathscr{J}^u$ so far, i.e. $\mathscr{J}^u = j_1^u \rightarrow j_2^u \rightarrow \ldots \rightarrow j_m^u$. Job history $\mathscr{J}^u$ can be used

to build a profile of user u, which expresses the skills u has learned in her history. This can be achieved by mapping each job $j_i^u \in \mathscr{J}^u$ performed by u to a cluster $c \in C$, computing the similarity between j_i^u and all clusters $\mathscr{C}$, and assigning to j_i^u the most similar cluster $c \in \mathscr{C}$. Since c is associated with a skillset $\mathscr{S}^c$, user u, who performed a job within cluster c, can be assumed to have acquired all the skills required by that job, i.e. $\mathscr{S}^c$. Iterating the same procedure over all past jobs $\mathscr{J}^u$, the user's profile $\mathscr{P}^u$ will be:

$$\mathscr{P}^u = \{(s_1^u, o(s_1^u, u)), (s_2^u, o(s_2^u, u)), \ldots, (s_z^u, o(s_z^u, u))\} \tag{11.3}$$

where $o(s,u)$ computes how much user u reinforced skill s. The o function simply counts the occurrences of s in the m skillsets $\mathscr{S}^{c_i}$ corresponding to the jobs $\mathscr{J}^u$ performed by u. One could object that a user who performed a job has not necessarily acquired every skill characterizing it. This is true, since it may depend on how many years the user has performed the job for, and possibly on other factors that can be taken into account when mapping user history to her skillset. However, these variables are beyond the scope of this work and will be disregarded for a matter of simplicity.

Rather than focusing on computing a user profile, we use user profiles as an input to our main contribution. Although the user's profile is conceptually different with respect to the states of the job graph, which are independent of the user, they can still be compared, since they are both expressed in terms of skills. The cluster wherein a candidate is most likely to get a job can be found by computing the similarity between candidate's skillset $\mathscr{P}^u$ and each cluster $c \in \mathscr{C}$. Similarity is obtained by summing the ranks of skills that are both in $\mathscr{P}^u$ and $\mathscr{S}^c$. For instance, the similarity between $\mathscr{P}^u = \{(\text{programmer}, 4), (\text{computer}, 2), (\text{technology}, 3)\}$ and $\mathscr{S}^c = \{\text{technology, computer, software}\}$ is equal to 5. The cluster with highest score is that wherein the candidate is most likely to get a job.

The last missing information to recommend a career pathway is the user's career goal. If the job graph was not large, the user could inspect it and directly choose the state she aims to reach, but this is not true in general, and may depend on the amount of data used to build the job graph. Moreover, in real world the user is more likely to provide a plain text description of her career goal, e.g. in her CV, or to clearly state which is the job she aims at. Under this assumption, the selection of the career goal state $c_g^u \in \mathscr{C}$ becomes identical to mapping a job to the most similar cluster. Therefore, the similarity between candidate's career goal g^u and all clusters $\mathscr{C}$ can be computed, selecting the most similar cluster $c_g^u \in \mathscr{C}$ as the goal state. Note that it may happen that the user's current state coincides with the goal state, in which case no career pathway is needed.

11.2.2 MDP recommender

Given a job graph G, a user's profile $\mathscr{P}^u$ and a goal state $c_g^u \in \mathscr{C}$ in the job graph, the career pathway recommendation problem can be formulated as a Markov Decision Process (MDP). An MDP is a four-tuple: $\langle S, A, R, T \rangle$, where S is a set of states, A is a set of actions, R is a reward function that assigns a real value to each action $a \in A$ taken from state $s_i \in S$ and ending in

state $s_j \in S$, T is a function that provides the transition probability between pair of states given an action. In an MDP, the decision-maker's goal is to behave so as to maximize the reward. An optimal policy is computed among all possible policies, and it corresponds to the one such that the reward will be maximized. In the CPR context, an optimal policy corresponds to the sequence of jobs that a user should try to get in order to reach her career goal.

It should be noted that both transition probabilities and rewards are broad concepts. Several factors may be taken into account for their definition, including salary, geographical constraints, and so on. Here we define them in terms of skills for simplicity, but other formulas can easily be plugged in. Let us see how the job graph, the user's profile, and the goal state in the job graph can be exploited to create an MDP. The user's profile $\mathscr{P}^u$ becomes a state $s_0 \in S$ of the MDP. The subscripts used for MDP state representation denote the job graph states wherein the user has got a job so far. For example, a user is in state $s_{\{0,4,8\}} \in S$ if she has got at least one job in cluster $c_4 \in \mathscr{C}$ and at least one job in cluster $c_8 \in \mathscr{C}$. When the user is in state $s_{\{0,4,8\}}$, she will be acquired both the skills in $\mathscr{S}^{c_4}$ and in $\mathscr{S}^{c_8}$, so the order in which she got the jobs is irrelevant. The job graph states $\mathscr{C}$ can be considered as the set of actions A available to the user. The user is free to apply for a job in whatever state of the job graph. Let us say that the user currently is in state $s_{Cur} \in S$, her profile is $\mathscr{P}^u_{Cur}$, and she takes an action $a_i \in A$, i.e. she applies for a job in cluster $c_i \in \mathscr{C}$. In case the application fails, neither the MDP state nor the user's profile will change. In case it succeeds, the user will reach a new career state $s_{Cur \cup i}$, and her profile will be updated according to the following formula:

$$\mathscr{P}^u_{Cur \cup i} = \mathscr{P}^u_{Cur} \cup \mathscr{S}^{c_i} \tag{11.4}$$

The rationale is that when the user gets a job in c_i, she will learn all the skills characterizing jobs in c_i. The skills already owned by the user will be strengthened, whereas new skills will be added to her profile. The success or failure of the application depends on the likelihood $l_{\mathscr{P}^u_{Cur} \to i} \in [0,1]$ for a user whose profile is $\mathscr{P}^u_{Cur}$ to get a job in cluster c_i, which is computed exactly as for the job graph. This means that for each state $s_{Cur} \in S$ and action $a_i \in A$, the transition probability to move to a state $s_{Next} \in S$ will be computed as follows:

$$t(s_{Cur}, a_i, s_{Next}) = \begin{cases} l_{\mathscr{P}^u_{Cur} \to i} & \text{if } s_{Next} = s_{Cur \cup i} \\ 1 - l_{\mathscr{P}^u_{Cur} \to i} & \text{if } s_{Next} = s_{Cur} \\ 0 & \text{otherwise} \end{cases} \tag{11.5}$$

The reward obtained by the user when she applies for a job in c_i depends on the application outcome and on the amount of skills required by the goal state c^u_g that u would acquire in case of a successful application. More precisely, let us define the benefit $b^g_{\mathscr{P}^u_{Cur} \to i} \in [0,1]$ for a user whose current profile is $\mathscr{P}^u_{Cur}$ to get a job in cluster c_i with respect to her career goal state c^u_g as follows:

$$b^g_{\mathscr{P}^u_{Cur} \to i} = 1 - \frac{|(\mathscr{S}^{c_g} \setminus \mathscr{P}^u_{Cur}) \setminus \mathscr{S}^{c_i}|}{|\mathscr{S}^{c_g} \setminus \mathscr{P}^u_{Cur}|} \tag{11.6}$$

The more skills c_i and c_g^u share among those that u does not own the higher the benefit for u to get a job in cluster c_i. Trivially, the benefit to get a job in the career goal state will be maximum, whereas the benefit to get a job in the current career state will be 0. The reward a user in state $s_{Cur} \in S$ will get when she lands in state $s_{Next} \in S$ after choosing action $a_i \in A$ can be computed as follows:

$$r(s_{Cur}, a_i, s_{Next}) = \begin{cases} 0 & \text{if } g \in s_{Cur} \\ M & \text{if } g \in s_{Next} \\ b^g_{\mathscr{P}^u_{Cur} \to i} & \text{if } s_{Next} = s_{Cur \cup i}, \\ & i \neq g, i \notin Cur \\ -\varepsilon & \text{if } s_{Next} = s_{Cur} \\ 0 & \text{otherwise} \end{cases} \tag{11.7}$$

where $s_g \in S$ represents the fact that the user has been able to reach her career goal state $c_g^u \in \mathscr{C}$, $M >> 1$ is a large positive integer, and $\varepsilon << 1$ is a small positive real. The rationale is that the reward will be very large, i.e. M, as soon as the user is able to reach her goal, thereafter no transition will be allowed. Otherwise, the reward depends on how much a new job will help the user reaching her career goal, e.g. how many skills required by the career goal and currently not owned by the user will be acquired through that job, or more generally, the reward can also reflect salary and other factors. The negative reward is given to avoid two undesirable behaviours of the user: applying for a job with a negligible likelihood to succeed, and applying for a job that has already been performed. Both behaviours are not beneficial in terms of career evolution, and will result in the user being stuck in the same career state. Trivially, the reward of unfeasible transitions is equal to 0.

The MDP process can be solved for example using the policy-iteration algorithm [How64], and an optimal policy can be extracted accordingly. The sequence of actions that maximizes reward corresponds to the recommended career steps for a user u whose initial profile is $\mathscr{P}^u$ and whose career goal is c_g^u. The skillsets $\mathscr{P}^u_{Cur}$ at each state corresponds to the evolution of the user's profile during the pathway.

11.3 Experimental evaluation

The experiments aim at validating to what extent the MDP-based approach is able to recommend a pathway to users having a well-defined career goal. The analysis is performed on a publicly available corpus, which has previously been employed for a Kaggle competition on job recommendation.[1] Since the original task is different from the purpose of this work, we use just a subset of the data, including job postings and users' job histories. The former comprise job title, description and requirements expressed in natural language, whereas the latter the sequence of jobs each user has performed. It should be noted that the users' past jobs are just names, such

[1] https://www.kaggle.com/c/job-recommendation/data

as *software engineer*, *software analyst*, *project manager*, and do not match with those in job postings.

The job postings are used to build the job graph. Standard text pre-processing techniques are applied, including tokenization, case folding, and lemmatization. Distributed representations of words and job postings are learned, relying on the gensim [RS10] implementation of paragraph vectors.[2] We use the distributed bag-of-words version of paragraph vectors, training word vectors in skip-gram fashion. The dimensionality of feature vectors is set to 100 after a tiny tuning, whereas the other parameters are left to their default values. Careful readers may find further details on parameters in the original papers [MCCD13, LM14] as well as in the previously linked gensim library. After having mapped the job postings into semantic-bearing feature vectors, Mini-batch K-means is applied to group similar jobs, empirically finding the best number of clusters, which is strongly dependent on data. The 30 most representative terms have been extracted from each cluster, and considered as the required skillset to get a job in it. Then the job graph is built as described in Section 11.1.2.

11.3.1 Evaluation protocol

Since the data are not crafted for career pathway recommendation, we design the following strategy to evaluate our approach. Each job in the user's history can be assigned to a cluster of the job graph, and the corresponding skillset can be extracted accordingly. The first job in the user's career, j_1^u, is used to build the user's initial profile as just described, whereas the last job, j_m^u, is considered as the career goal. Our MDP-based approach will recommend a pathway from the initial profile to the career goal, which can be compared with the natural path the user walked (i.e. her sequence of jobs) in order to assess the goodness of our method.

Evaluation metrics should take into account to what extent the career path recommender can be useful to the user in a real scenario. Indeed, a user might want to use the system 1) if it is able to recommend a pathway towards her career goal, 2) if the recommended path is shorter than the one she would walk by herself (i.e. less time is required to reach the goal), and 3) if the recommender is able to satisfy as many desirable side conditions as possible during the pathway, such as taking into account salary, location and other constraints.

In our evaluation protocol, the first point is satisfied by hypothesis by the user pathway, since we are committed to consider the last job as the career goal. If the career goal was a future job the user is aiming at, which is the most frequent use case, the user would not be guaranteed to be able to reach the goal. On the other hand, the ability of the recommender to reach the goal state mainly depends on the goodness of the job graph. The more the job graph represents a network of jobs, the more the MDP-based approach is able to recommend a pathway towards the goal. For evaluation purposes, we use different number of clusters to build the job graph, assessing the accuracy accordingly, which is the fraction of times it is able to lead the user towards her goal. The second point is very important, because nobody would use a system that recommends

[2] https://radimrehurek.com/gensim/models/doc2vec.html

a pathway that takes forever to reach the goal. The system will be useful if the number of hops required to reach the goal state through the recommended policy is lower than that required by the user. More formally, let $\mathscr{R}^u$ be the sequence of future jobs that will be recommended to user u, let U be the number of users, and let $\mathscr{G}$ be the set of good recommendations. The job history $\mathscr{J}^u$ of user u can be used as simulation of the future evolution. As previously explained, the first job j_1^u of the sequence $\mathscr{J}^u$ is used to extract the user's profile, the last job j_m^u is interpreted as the career goal, whereas the intermediate jobs $j_2^u \rightarrow j_3^u \rightarrow \ldots \rightarrow j_{m-1}^u$ represent the pathway walked by the user towards the goal. $\mathscr{J}^u$ can be compared with $\mathscr{R}^u$ to extract $\mathscr{G}$. In particular, a recommendation can be considered to be good if $|\mathscr{R}^u| < |\mathscr{J}^u|$ and the last job of the sequence $\mathscr{R}^u$ is the career goal. The metric based on the number of hops, NHM, will be a value in $[0,1]$ computed as follows:

$$NHM = \frac{|\mathscr{G}|}{U} \tag{11.8}$$

In case the duration of jobs was available, it would be worthwhile that the whole pathway duration was minimized rather than the number of hops, and the metric should be adapted accordingly. The third point mentioned above is beyond the scope of this paper, because further constraints have not been considered for the sake of keeping the approach as simple as possible. If constraints were added, it could also be worthy to assess whether the system is able to meet them, although this is not the primary goal of a career pathway recommender.

11.3.2 Results

Figure 11.2 shows the *Utility* of the recommender, which is the ratio between NHM and the accuracy of the recommender. The *Utility* represents the fraction of times the recommender suggests a shorter pathway to the user, given that it is able to recommend a pathway that eventually reaches the goal. Computing the *Utility* over all users of the analyzed corpus, it is always over 0.8. Considering that the average pathway length of the analyzed users is 4.23, the outcome is particularly valuable, because it is hard to optimize an already short pathway. The *Utility* is almost constant by increasing the number of clusters used to build the job graph, but some deviations may occur depending on the likelihood to move from one cluster to another, which affects the optimal policy computed by the MDP-based recommender. Since in our simulation, for a matter of simplicity, the likelihood to get a job in a cluster only depends on the overlapping factor, results can either positively or negatively be affected by variations in the job graph structure.

The second experiment we performed only takes into account the users whose pathway length is at least 10, so as to evaluate the performance of the MDP-based recommender in reaching long term goals. The results in Figure 11.3 show that the accuracy of the recommender and NHM generally grow with the number of clusters. When few clusters are employed, the job graph is not appropriately represented and the recommender struggles to identify the career goal. Increasing the number of clusters, performance improves because jobs are better represented and it is easier

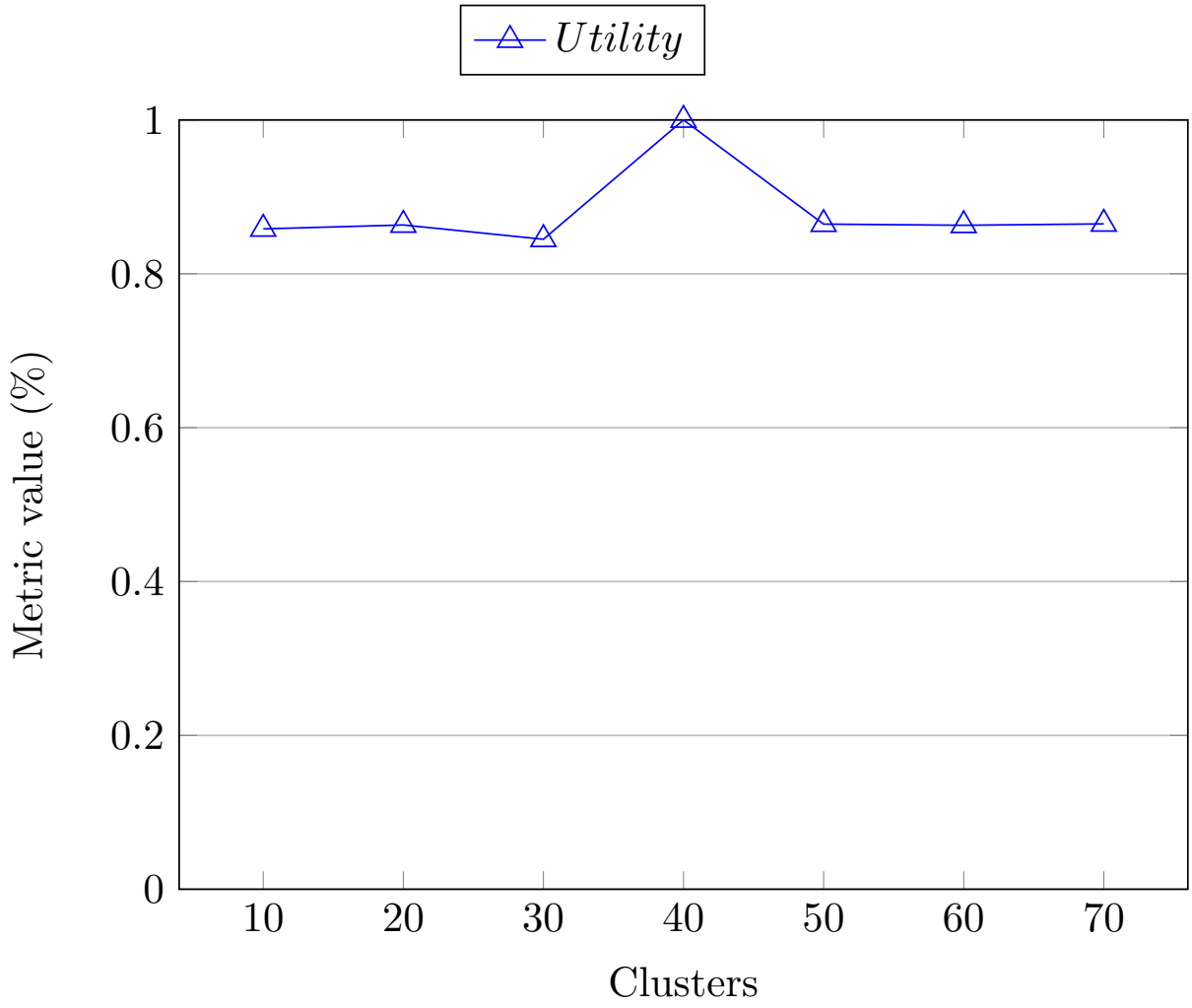


Figure 11.2: The *Utility* of our system varying the number of clusters employed to build the job graph. *Source: [PAB+].*

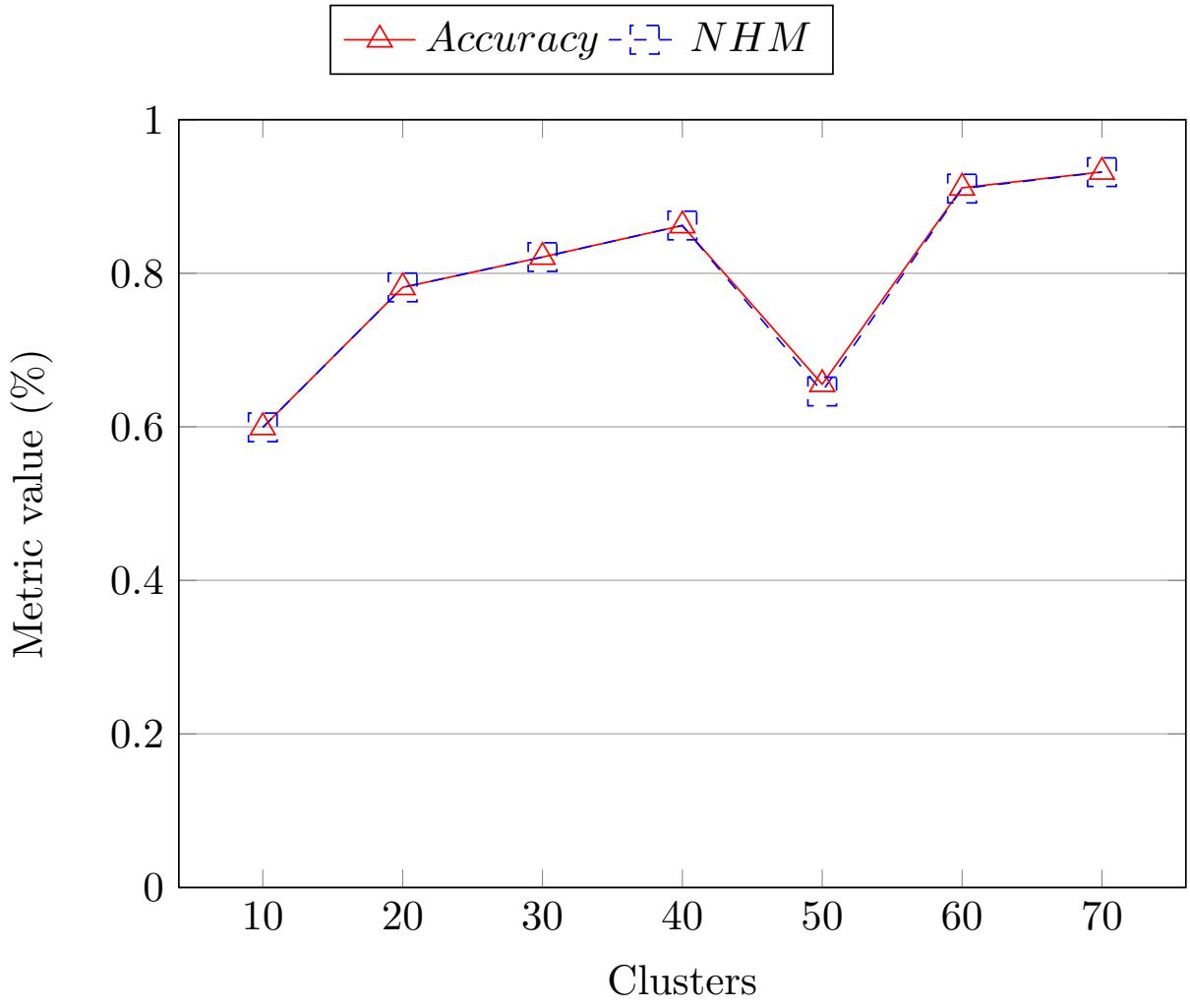


Figure 11.3: The accuracy and *NHM* of our system for users who performed at least 10 jobs. *Source: [PAB+].*

to recognize the goal. However, readers may have noted that accuracy decreases when the job graph is constructed with 50 clusters. This outcome depends again on the way we computed likelihood in the simulation. Relying for simplicity on the overlapping factor to compute likelihood and setting l_{thr} to 0.2 to avoid unlikely transitions and speed up computation, some connections between states can possibly be removed, and the recommender can be prevented from reaching the career goal. In a real world scenario, where the likelihood also depends on other constraints, such as for example salary and location, the MDP-based recommender will benefit from larger number of clusters. Careful readers may have noted that when the recommended pathway succeeds to reach the goal, it is almost always shorter than the pathway walked by the user. Indeed, accuracy and NHM overlap regardless of the number of clusters, meaning that the *Utility* always approaches 1. The outcome proves the ability of the MDP-based recommender in reaching long term goals, recommending a pathway that is both safer and shorter than that the user would walk.

In a real world scenario, it is advisable to choose a number of clusters that maximizes NHM. This means that the system is able to recommend a shorter pathway towards the goal than the user is able to do, which is even safer because all possible pathways have been explored by the MDP. The right number of cluster can be established by following the evaluation protocol we presented here, then a pathway towards a future career goal can be recommended. Anyway it is worthwhile to remember that for each user a different MDP has to be built, and the possible MDP states are $2^{|\mathscr{C}|}$ to take into account the possible career evolutions. For example, with 10 clusters we have to build and solve an MDP with $2^{10} \approx 10^3$ states for each user, with 20 clusters the MDP has $2^{20} \approx 10^6$ states, with 30 clusters $2^{30} \approx 10^9$ states, and so on. Therefore, the right choice may also depend on computational constraints.

11.4 Final remarks

This chapter has introduced a skillset-based approach to model job transitions and build a job graph, which is independent of information about candidates. The job graph is then used in combination with a candidate's job history and career goal to create a Markov Decision Process, whose solution provides the candidate with an optimal policy (according to a model built for the problem) to follow in order to eventually achieve her career goal.

MDPs are suitable for recommending a career pathway, because they are able to model uncertainty and rewards, both at the goal state and along the way, dealing with unexpected situations. Similar jobs are mined by a clustering of job postings, and the most relevant terms of each cluster are used to express the likelihood to move from one job cluster to another. The MDP-based method can easily be customized to comply with changing requirements or to take advantage of further available information. Beyond providing a recommendation inherently aware of contingencies in career evolutions, our method helps users achieving their career goal in an optimized manner. The recommended pathway is generally shorter than the one the user would walk. Also, the method is flexible to define optimality, e.g. it could involve minimizing

the intermediate career steps, or strengthening the user's profile recommending intermediate career steps so that the user can learn skills required by her career goal.

Part V

Accelerate the Development of Big Data Analytics

12 Towards vendor-agnostic implementation of big data analytics

In the age of Big Data, scalable algorithm implementations as well as powerful computational resources are required, as previously discussed in Chapters 2 and 6. For data mining and data analytics the support of big data platforms is becoming increasingly important, since they provide algorithm implementations with the resources needed for their execution. However, choosing the best platform might depend on several constraints, including but not limited to computational resources, storage resources, target tasks, service costs. Sometimes it may be necessary to switch from one platform to another depending on the constraints. As a consequence, it is desirable to reuse as much algorithm code as possible, so as to simplify the setup in new target platforms. Unfortunately each big data platform has its own peculiarity, especially to deal with parallelism. This impacts on algorithm implementation, which generally needs to be modified before being executed.

This chapter introduces functional parallel primitives to define the parallelizable parts of algorithms in a uniform way, independent of the target platform. Primitives are then transformed by a compiler into skeletons, which are finally deployed on vendor-dependent frameworks. The procedure proposed aids not only in terms of code reuse but also in terms of parallelization, because programmer's expertise is not demanded. Indeed, it is the compiler that entirely manages and optimizes algorithm parallelization.

12.1 Background

Under the explosive increase of global data, the term "big data" is mainly used to describe enormous datasets. Big data typically includes masses of unstructured data that need more efficient analysis in comparison with traditional datasets [CML14]. Among the four phases of the big data value chain, which includes data generation, data acquisition, data storage, and data analysis, this chapter focuses on the last one. Big data analysis consists in designing appropriate mining algorithms to find useful insights from big data [TLCV15]. Data mining algorithms are pivotal for big data analysis in terms of computational cost, memory requirement, and accuracy of results. They are generally tailored to specific problems, such as clustering [FAT+14, XLGC12],

classification [TvdS13, RML14] and frequent pattern mining [LLH12, LMJ14]. Differently from the data mining design for specific problems, machine learning algorithms can be used for different mining and analysis problems, because they are typically employed as the search algorithm of the required solution [WFHP16]. Since most machine learning algorithms can be used to find an approximate solution for optimization problems, they can be employed for those data analysis problems that can be formulated as optimization problems.

Machine learning algorithms need to be scalable in order to deal with large amount of data. Furthermore, powerful computational resources are generally required. The support of big data analytics platforms is therefore advisable to provide whatever is needed for algorithm execution. Many platforms have been developed so far to provide such a support, as already stated in Section 6.2. Apache Hadoop [Had09] is an open source framework for storing and processing large datasets using clusters of commodity hardware. Hadoop is designed to scale up to hundreds of nodes, and it also is fault tolerant. The programming model used in Hadoop is MapReduce [DG08]. Some wrappers have been developed for MapReduce to foster the source code development, including Apache Pig by Yahoo! [ORS+08] and Hive by Facebook [TSJ+09]. One of the major drawbacks of MapReduce is its inefficiency in running iterative algorithms, because MapReduce is not designed for iterative processes. Spark [ZCF+10] is a next generation paradigm for big data processing. Spark is able to perform in-memory computations, so as to allow data to be cached in memory, eliminating the Hadoop's disk overhead limitation for iterative tasks. A remarkable alternative to Spark is Stratosphere [ABE+14], an open-source software stack for parallel data analysis that brings together a unique set of features that allow expressive, easy, and efficient programming of analytical applications at very large scale.

Careful readers may find further details on Big data mining platforms and algorithms in Chapter 6.

12.2 Parallel programming primitives

This section introduces parallel primitives that support non-expert programmers in the definition and implementation of vendor-agnostic big data mining algorithms, preventing platform lockdowns. Each primitive represents a highly abstract parallelization paradigm, which can be used in the parallelizable sections of agnostic code. The primitives will then be processed by a compiler in order to be deployed on specific target platforms, as will be shown in Section 12.4.

The idea is that code can be reused among platforms and that programmers do not have to take care of optimizing the parallel code fragments. Programmers should just identify the parallelizable fragments of code and manage high-level parallel primitives, whereas the choice of a specific lower-level paradigm, the code conversion into vendor-specific target environments and the optimization of parameters are handled by the compiler.

The primitive-based implementation of several algorithms is shown below, including clustering, classification, association rule ming and page ranking techniques.

12.2.1 Primitives definition

Three high-level parallel primitives are shown, called *data_parallel_region*, *data_parallel_region_with_reduction* and *task_independent_region*. These primitives cover the parallel paradigms currently supported by the compiler, but others can easily be introduced as the compiler will be extended.

The first parallel primitive, whose signature is

$$data\_parallel\_region(items, func, *args)$$

returns a collection with the objects in *items* transformed according to the function *func*, which returns the transformed version of the object given as input. An arbitrary number of arguments, *args*, can additionally be passed to *func*. In a sequential execution, the result is equivalent to a list comprehension, a construct used to encapsulate actions which must be executed on each item of a collection. In a parallel execution, this permits to distribute the computation of an operation, applying it on different, distributed subsets of the input data. Basically, this function allows to exploit parallelism for operations that perform one-to-one transformation of objects.

Listing 12.1: Sequential implementation of data_parallel_region.

```
def data_parallel_region(data, func, *args):
  return [func(x, *args) for x in items]
```

In many cases where *data_parallel_region* is used, additional operations are also required to aggregate the items of the resulting collection into a single one, according to some specific logic. To address this need, a second parallel primitive is introduced, *data_parallel_region_with_reduction*, whose signature is

$$data\_parallel\_region\_with\_reduction(items, func, reduce\_func, *args)$$

That is logically equivalent to the direct application of an aggregation function to the results of a *data_parallel_region* call on a function *func*. The *data_parallel_region_with_reduction* primitive is defined as follows:

Listing 12.2: Sequential implementation of data_parallel_region_with_reduction.

```
def data_parallel_region_with_reduction(items, func, reduce_func, *
    args):
  return reduce_func([func(x, *args) for x in items])
```

The last parallel primitive presented is *task_independent_region*, whose signature is

$$task\_independent\_region(data, func\_list, arg\_list)$$

It applies a list of functions to the input data, and returns a collection containing the results of the single functions. A list of arguments is included if required by some of the functions. This primitive allows distributing the computation of such functions, so as to apply them on the input data simultaneously.

Listing 12.3: Sequential implementation of task_independent_region.

```
def task_independent_region(data, func_list, arg_list):
  result = {}
  length = len(funcList)-1
  for i in range(length):
    if params[i]:
      result[func_list[i].__name__]=func_list[i](data, params[i])
    else:
      result[func_list[i].__name__]=func_list[i](data)
  return result
```

12.3 Agnostic implementations of data mining algorithms based on parallel primitives

This section shows some examples of well-known data mining algorithms, implemented using the parallel primitives defined in Section 12.2.1. The algorithms that will be shown below are the following:

k-means is a partitioning clustering technique, which is able to split the instances of a data set into k non-overlapping groups, minimizing intra-cluster distance as well as maximizing inter-cluster distance [MAC67, Llo82]. The algorithm basically samples k initial centroids, either randomly or by means of a heuristic technique. Each point of the dataset is assigned to one of the k partitions defined by the centroids. Then, the centroid of each partition is computed, according to the points assigned to it. Afterwards, k new partitions are created, and the procedure iterates until the algorithm eventually converges. The algorithm is guaranteed to converge in a finite number of steps.

C4.5 is an algorithm for supervised learning of decision tree-based classifiers [Qui93]. Starting from a root, each intermediate node of the decision tree points to other nodes according to the value of a single feature, until a leaf node indicating the instance class is eventually reached. C4.5 algorithm works by selecting the single feature within training data that better discriminates instances of different classes; it then creates a node based on such a feature, splits the dataset into groups according to its value, and recursively runs the same procedure on these groups.

Apriori is a data mining algorithm for frequent itemset finding over transactional databases [AS$^+$94]. Basically, each transaction is seen as an itemset. Given a threshold *thr*, it finds the item sets that are subsets of at least *thr* transactions in the database. The algorithm iteratively generates candidates, i.e. it extends subsets one item at a time, and tests groups of candidates against the data. The process goes on while new candidates can be generated.

The frequent itemsets determined by Apriori can be used for applications as market basket analysis, whose goal is finding association rules in the database that highlight some trends.

PageRank is an algorithm for ranking pages of a hypertext, according to the number and importance of referring pages. The algorithm was designed to statically rank web pages using their hyperlink structure only, thus it is independent of search queries [BP98]. The set of hyperlinked pages is modeled as a graph (V,E), where each vertex $v \in V$ is a page and E is the binary relation

$$\{(u,v) \in V \times V : \text{page } u \text{ contains a hyperlink to page } v\}.$$

The rank P_v of a page v depends on the sum of weighted votes from referring pages, in which each referring page u has exactly one weighted vote. The weight of u's vote is computed as page rank P_u divided by the number of referrals O_u on the page. The sum of votes is multiplied by a damping factor d and summed to its unit complement divided by the number of pages:

$$P_v = \frac{1-d}{|V|} + d \sum_{(u,v)\in E} \frac{P_u}{O_u} \qquad v \in V \tag{12.1}$$

yielding an eigensystem of equations in the variables P_v, $v \in V$. To solve Equation (12.1), PageRank computes the ranking vector P until the L^1-norm of the difference between consecutive iterations is smaller than a predefined threshold thr.

12.3.1 k-means

K-means algorithm has a relatively simple procedural description. As discussed above, the main loop of the algorithm entails assigning each data point to a cluster and computing new cluster centres from such assignments: these two steps can be straightforwardly defined by few primitives, where input data are the points to be clustered.

Given a dictionary C that maps cluster keys to respective centres, the $\textsc{Assign}(C)(i,x)$ function must associate to each data point x the identifier of its cluster.

$$\textsc{Assign}(C) = x \mapsto \{(x, \underbrace{\operatorname{argmin}_{c \in \text{keys}(C)} \text{Distance}(x, C(c))}_{\text{key of cluster with nearest centre}})\} \tag{12.2}$$

Then, we create a set G of clusters, each constituted by an identifier and a set of points belonging to it. For each of them, the $\textsc{Centroid}(c, A_c)$ function computes its new centre.

$$\textsc{Centroid}(X_c) = \{(c, \frac{1}{|X_c|} \sum_{x \in X_c} x)\} \tag{12.3}$$

K-means algorithm can then be defined as the iterative application of these functions, with the addition of a procedure for the initialization of centroids and the verification at each step of the termination conditions.

Algorithm 1 Pseudocode definition of k-means algorithm.

procedure KMEANS(X, k)
  $C_0 \leftarrow$ INITCENTROIDS(k, x)
  $i \leftarrow 0$
  while $i = 0 \vee C_i \neq C_{i-1}$ **do**
    $A_i \leftarrow \{\text{ASSIGN}(C_i, x) : x \in X\}$
    $\forall c : G_i(c) \leftarrow \{x : (c, x) \in A_i\}$
    $\forall c : C_i(c) \leftarrow \text{CENTROID}(G_i(c))$
    $i \leftarrow i + 1$
  return C_i

Listing 12.4: Vendor-agnostic definition of k-means.

```
import math
import random

def distance(a, b):
  """Computes euclidean distance between two vectors"""
  return math.sqrt(sum([(x[1]-x[0])**2 for x in zip(a,b)]))

def kmeans_init(data, k):
  """Returns initial centroids configuration"""
  return dict(enumerate(random.sample(data, k)))

def kmeans_assign(p, centroids):
  """Returns the given instance paired to the key of the nearest
     centroid"""
  comparator = lambda x: distance(x[1], p)
  nearest = min(centroids.items(), key=comparator)
  return {nearest[0]: (1, p)}

def kmeans_aggregate(aggrs):
  """Aggregates partial vector sums to compute centroids"""
  totals = {}
  for aggr in aggrs:
    for (c, (nb, sb)) in aggr.items():
      if c in totals:
        (na, sa) = totals[c]
        totals[c] = (na+nb, map(sum, zip(sa, sb)))
      else:
        totals[c] = (nb, sb)
  return totals
```

```
def kmeans_centroid(cluster_point, cluster_size):
  """Returns the centroid (mean point) of the given instances"""
  return cluster_point/cluster_size

def kmeans(data, k):
  """Clusters data in k groups and returns their centroids"""
  centroids = kmeans_init(data, k)
  iteration = 0
  while iteration == 0 or previousCentroids != centroids:
    previousCentroids = centroids
    sums = data_parallel_region_with_reduction(data, kmeans_assign,
       kmeans_aggregate, centroids)
    centroids = {}
    for x in sums.items():
      centroids[x[0]] = data_parallel_region(x[1][1], kmeans_centroid
         , x[1][0])
    iteration += 1
  return centroids
```

12.3.2 C4.5

C4.5 algorithm can be expressed in a recursive divide-and-conquer fashion: for each non-leaf node of the tree, the dataset is split into disjoint and exhaustive groups according to the value of a feature. The same procedure is then applied to such groups to obtain an equal number of subtrees. The construction of disjoint subtrees can proceed in parallel.

For each group, starting from the whole dataset corresponding to the root node, the branching feature must be defined. In order to pick the one yielding the most homogeneous groups in terms of class labels, we compare them by means of information gain.

Afterwards, we need to define the groups into which the dataset of the current node is split: such groups correspond to the possible values of the selected feature. While a discrete feature has a finite number of possible values, for continuous features a small number of ranges covering the set of possible values needs to be selected, and branches have to be created accordingly. In either case, this operation can be encapsulated in a SELECTRANGES function.

For each selected range, the subset of instances having the value of the branching feature within that range is selected, and the whole procedure is applied on it recursively. Single subtrees yielded by each call are finally merged as branches of a single node, returned by the procedure.

The algorithm termination is guaranteed by checking for base cases at the algorithm start. If all current instances are labeled with the same class, a leaf pointing to that class is returned.

Algorithm 2 Pseudocode definition of C4.5 algorithm.

procedure C4.5(X)
  if $\exists k : c(x) = k \forall x \in X$ **then**
    return LEAF(k)
  bestFeature $\leftarrow$ argmax$_{f \in \text{features}(X)}$ INFORMATIONGAIN(X, f)
  $R \leftarrow$ SELECTRANGES(bestFeature,X)
  branches $\leftarrow \emptyset$
  for all r in R **do**
    $S_v \leftarrow \{x \in X : x[\text{bestFeature}] \in r\}$
    $T_v \leftarrow$ C4.5(S_v)
    branches $\leftarrow$ branches $\cup \{(v, T_v)\}$
  return COMPOSITE(bestFeature,branches)

Listing 12.5: Vendor-agnostic definition of C4.5

```
class TreeLeaf:
  """Leaf of a decision tree"""
  def __init__(self, label):
    self.label = label
  def classify(self, data):
    return self.label

class TreeNode:
  """Internal node of a decision tree"""
  def __init__(self, feature, branches):
    self.feature = feature
    self.branches = branches
  def classify(self, data):
    for (values, subtree) in self.branches:
      if data[self.feature] in values:
        return subtree.classify(data)
    else:
      raise Exception('unexpected value')

def ranges(data, feature): # discrete values are assumed
  """Returns ranges of possible values of the given feature"""
  return [[x] for x in set([d[feature] for d in data])]

def infogain(data, feature, target):
  """Computes information gain on the given feature"""
  # omitted for brevity
```

```
def c45_select(features, target, data):
  """Selects the feature to be used for branching"""
  featGains = [(f, infogain(data, f, target)) for f in features]
  return max(featGains, key=lambda x: x[1])[0]

def c45_branch(data, features, target):
  """Constructs a single branch"""
  return (data[0], c45_subtree(features, target, data[1]))

def c45_subtree(features, target, data):
  """Constructs a decision tree from the given data"""
  # test whether all instances have the same class label
  onlyLabel = data[0][target]
  for x in data[1:]:
    if x[target] != onlyLabel:
      break
  else:
    return TreeLeaf(onlyLabel)
  # otherwise, split instances on the best feature
  f = c45_select(features, target, data)
  subdata = [(r, []) for r in ranges(data, f)]
  for x in data:
    for (r, s) in subdata:
      if x[f] in r:
        s.append(x)
  branches = data_parallel_region(subdata, c45_branch, features,
     target)
  return TreeNode(f, branches)
```

12.3.3 Apriori

In Apriori algorithm, apart from the initialization procedure INIT1ITEMSETS, which is tailored to find the frequent 1-itemsets, three main functions can be identified to carry out frequent itemsets finding, candidates generation and subsets finding.

Given a set C_k of candidate itemsets and a support threshold thr, the FINDFREQUENTITEMSETS function finds the frequent itemsets among the candidates, namely those whose support overcomes the threshold.

$$\text{FINDFREQUENTITEMSETS} = (C_k, thr) \mapsto \{c \mid c \in C_k \wedge count[c] \geq thr\} \tag{12.4}$$

The GENERATECANDIDATES function generates the candidate frequent itemsets for step k, given

the frequent itemsets for step $k-1$.

$$\begin{aligned}\text{GENERATECANDIDATES} = (L_{k-1}) \mapsto \{a \cup \{b\} \mid a \in L_{k-1} \wedge b \notin a\} \\ -\{c \mid \{s \mid s \subseteq c \wedge |s| = k-1\} \not\subseteq L_{k-1}\}\end{aligned} \tag{12.5}$$

Finally, the INCREMENTSUBSETSSUPPORT function finds all the candidate itemsets that are in the transactional database X and increments their support.

Algorithm 3 Pseudocode of the INCREMENTSUBSETSSUPPORT function.

procedure INCREMENTSUBSETSSUPPORT(C_k, X)
  for all $x \in X$ **do**
    $C_x \leftarrow \{c \mid c \in C_k \wedge c \subseteq x\}$
    for all $c \in C_x$ **do**
      count$[c] \leftarrow$ count$[c] + 1$

Algorithm 4 Pseudocode definition of Apriori algorithm.

procedure APRIORI(X, thr)
  $C_1 \leftarrow$ INIT1ITEMSETS(X)
  $L_1 \leftarrow$ FINDFREQUENTITEMSETS(C_1, thr)
  $k \leftarrow 2$
  while $L_{k-1} \neq \emptyset$ **do**
    $C_k \leftarrow$ GENERATECANDIDATES(L_{k-1})
    INCREMENTSUBSETSSUPPORT(C_k, X)
    $L_k \leftarrow$ FINDFREQUENTSUBSETS(C_k, thr)
    $k \leftarrow k + 1$
  return $\bigcup_k L_k$

Listing 12.6: Procedural definition of Apriori

```
import itertools

def combinations(items, size):
  return [frozenset(x) for x in itertools.combinations(items, size)]

def apriori_candidates(allitems, n, sets):
  """Generates candidate n-itemsets from (n-1)-itemsets"""
  return {x: 0 for x in [a.union([b]) for a in sets for b in allitems
      if b not in a] if all([(frozenset(c) in sets) for c in
     combinations(x, n-1)])}

def apriori_initcounts(tx, n, candidates):
```

```
  """Counts occurrences of the candidate itemsets in data"""
  return {s: 1 for s in combinations(tx, n) if s in candidates}

def apriori_sumcounts(counts):
  """Sums occurrences of itemsets"""
  total = {}
  for p in counts:
    for c in p:
      if c in total:
        total[c] += p[c]
      else:
        total[c] = p[c]
  return total

def apriori_frequent(candidates, thr):
  """Returns candidates with a minimum frequency"""
  return [c for (c, n) in candidates.items() if n >= thr]

def apriori(data, thr):
  """Returns frequent itemsets among given data"""
  allitems = frozenset([x for t in data for x in t])
  candidates = [frozenset([x]) for x in allitems]
  counts = data_parallel_region_with_reduction(data,
     apriori_initcounts, apriori_sumcounts, 1, candidates)
  frequent = apriori_frequent(counts, thr)
  result = frequent
  k = 2
  while len(frequent) > 0:
    candidates = apriori_candidates(allitems, k, frequent)
    counts = data_parallel_region_with_reduction(data,
       apriori_initcounts, apriori_sumcounts, k, candidates)
    frequent = apriori_frequent(counts, thr)
    result.extend(frequent)
    k += 1
  return result
```

12.3.4 PageRank

PageRank consists of an initialization phase, which computes outdegrees for, and assigns equal initial ranks to all pages, and a main loop, which computes the power iteration that solves the eigensystem by recomputing new values for all ranks, using Equation (12.1). Algorithm 5 is a pseudocode definition of PageRank. Outdegree computation is performed by the OUTDEGREES

Algorithm 5 Pseudocode definition of PageRank algorithm.

procedure PAGERANK(V, E, thr, d)
  $O \leftarrow$ OUTDEGREES(V, E)
  $P \leftarrow \{\langle v, |V|^{-1}\rangle : v \in V\}$
  $k \leftarrow 1$
  while $k = 1 \vee$ L1NORM$(P', P) \geq thr$ **do**
    $W \leftarrow$ INWEIGHTS(E, P, O)
    $P' \leftarrow P$
    $P \leftarrow \{\langle v, (1-d)/|V|\rangle : \nexists u (u, v) \in E\} \cup \{\langle v, (1-d)/|V| + dw\rangle : \langle v, w\rangle \in W\}$
    $k \leftarrow k + 1$

function, defined by

$$\text{OUTDEGREES} = (V, E) \mapsto \{\langle v, c\rangle : c = |\{\langle v, u\rangle : (v, u) \in E\}|, v \in V\}. \tag{12.6}$$

Ranking P is initialized straightforwardly by assigning equal values $1/|V|$ to all pages. The main loop iterates as long as the L^1-norm difference, computed as

$$\text{L1NORM} = (P', P) \mapsto \sum_{\substack{\langle v, r\rangle \in P \\ \langle v, r'\rangle \in P'}} |r' - r|, \tag{12.7}$$

is larger than thr. In the loop body, aggregate weights of incoming edges are computed by the following function:

$$\begin{aligned} &\text{INWEIGHTS} = \\ &\quad \langle E, P, O\rangle \mapsto \{\langle v, w\rangle : w = |\{\langle v, r/c\rangle : \langle u, r\rangle \in P, \langle u, c\rangle \in O, (u, v) \in E\}|\}. \end{aligned} \tag{12.8}$$

Current ranks are then saved to compute the loop condition. Updated ranks are obtained as a linear combination of unity and the weights above, assuming zero weight for pages having no ingoing edge.

Listing 12.7 is a procedural definition of PageRank. Outdegrees are computed first by associating a unit count with a page for each edge originating in the page, then by grouping all unit values associated with the same page, and finally by summing the units for each page. The rank dictionary is initialized as a sequence of $|V|^{-1}$ values, where $|V|$ is the number of pages. In the main loop, ingoing edges are processed. For each edge, its weighted vote is computed as the ratio between the current rank of the referring page and its outdegree associated with the destination edge, thereby mapping $\lambda e.\langle e_1, P_{e_0}/O_{e_0}\rangle)$ to the edge list E. Then, a sum reduction is applied to group weighted votes by page. Finally, the new ranks of the current iteration are computed: the damping factor d is multiplied by the votes, and its complement is added by mapping $\lambda v.\langle v_0, (1-d)/|V| + dv_1\rangle$ to the output of the last reduction. Note that the resulting pairs only involve pages which have some ingoing edge; the remaining pages should take the

rank $(1-d)/|V|$ by Equation (12.1). Therefore, the rank dictionary is reinitialized with values $(1-d)/|V|$, then updated using the new ranks. The last lines of the procedure recompute the L^1-norm of the residual and update the rank dictionary.

Listing 12.7: Procedural definition of PageRank

```
def sum_by_key(pairs):
  totals = {}
  for (k, v) in pairs:
    if k in totals:
      totals[k] += v
    else:
      totals[k] = v
  return totals

def pagerank_votes(edge, p, outdegrees):
  return (edge[1], p[edge[0]] / outdegrees[edge[0]])

def pagerank_newp(v, n, d, aggr_votes):
  if v in aggr_votes:
    return (v, (1-d)/n+d*aggr_votes[v])
  else:
    return (v, (1-d)/n)

def first_lambda(x):
  return (x[0],1)

def second_lambda(v, p, new_p):
  return abs(p[v]-new_p[v])

def custom_sum(lst):
  accumulator = 0
  for element in lst:
    accumulator += element
  return accumulator

def pagerank(vert, g, thr, d):
  """Return the page rank of vertices of graph (vert, g)."""
  n = len(vert)
  outdegrees = data_parallel_region_with_reduction(g, first_lambda,
     sum_by_key)
  p = {v: 1/n for v in vert}
  k = 1
  while k == 1 or l1_norm_diff >= thr:
    aggr_votes = data_parallel_region_with_reduction(g,
```

```
      pagerank_votes, sum_by_key, p, outdegrees)
   new_p = dict(data_parallel_region(vert, pagerank_newp, n, d,
      aggr_votes))
   l1_norm_diff = data_parallel_region_with_reduction(vert,
      second_lambda, custom_sum, p, new_p)
   p = new_p
   k = k + 1
return p
```

12.4 From vendor-agnostic implementation to vendor-specific platforms

This section describes an approach to compile the vendor-agnostic parallel programming primitives introduced in Section 12.2 into vendor-specific platforms. Basically, a compiler takes as input the algorithm definition based on the parallel primitives, processes it, and produces the necessary execution and deployment skeletons according to a suitable parallel computational pattern. Parallel computational patterns derive from the general design pattern definition. They represent reusable and portable solutions to parallelization problems, where computation and the data required for their execution need to be distributed among computing nodes.

12.4.1 Compiling primitives into skeletons

The incarnation of skeletons is addressed by a code compiler, developed following a code-based approach and introduced in [DMDE17]. In order to deploy an application on a target environment, the programmer either rewrites or develops from scratch her code starting from a set of parallel primitives, as those presented in Section 12.2.1. Such primitives are then mapped to a set of skeletons according to a suitable parallel pattern, which are filled by the skeleton-based compiler according to a set of pre-defined rules. In particular, the compiler is composed of:

- a *parser* to produce the algorithm's Abstract Syntax Tree (AST)
- a *skeleton filler* to alter the AST according to the primitives and the derived parallelization paradigms
- an *unparser* to produce the output skeleton
- a *set of rules*, bound to the chosen pattern, guiding the skeleton filling

The parser and the skeleton filler represent the executive core of the compiler. The parser takes the algorithm written in Python as input, and generates its abstract syntax tree representation. The tree nodes represent the key constructs of the chosen programming language, and are used

to define the rules that are at the base of the compiler. The algorithm is annotated with micro-functions that are recognized by the parser and that trigger a separate set of rules. The rules depend not only on the specific functions encountered, but also on the target pattern used for the execution skeletons, and on the target services/platforms for which appropriate deployment templates need to be constructed.

On the other hand, the skeleton filler deals with the automatic transformation of the syntactic tree, according to the triggered rules. The filler transforms the original syntax tree by moving, rearranging or creating nodes, which are then appended to a new tree. At the end, the produced AST is ready to be unparsed in order to obtain the desired skeletons. The skeletons produced by the filler are grouped into three categories:

Main Scripts The main scripts represent the entry point of the execution process managed by the skeleton. They contain the sequential code that cannot be distributed, and the calls to the secondary scripts. The libraries required to execute the sequential code and to call the secondary scripts are also imported at the beginning of the main scripts. If the sequential code contains calls to locally defined functions or procedures, they need to be declared in this script.

Secondary Scripts The secondary scripts contain the code that will be executed in parallel on different computational nodes. As for the main scripts, the required libraries and function declarations are reported at the beginning. The number of produced scripts depends on the selected pattern.

Deployment Templates The deployment templates contain information on the characteristics of the computational nodes the filled skeletons will be executed on. Such characteristics are derived from the declarative model or are left to the user to be completed.

The skeleton filling rules represent the knowledge base on which the compiler works to enact the transformations of the original code. The rules are pattern specific, as different patterns require different transformations and secondary scripts. However, they are completely independent of the algorithm, as the parser treats the same micro-function in the same way, whichever is the algorithm wherein they are used. This means that we have specific rules for the implementations of MapReduce, BagOfTasks, Producer-Consumer, and so on.

Figure 12.1 provides a step-by-step description of the rules enforced by the compiler when a *data_parallel_region* primitive is identified and the MapReduce pattern is applied. Each of the steps composing the diagram represents a transformation occurring to the AST of an empty skeleton, which is filled by moving nodes from the AST representation of the original sequential code, annotated with the primitive. As an instance, consider the step highlighted in Figure 12.2. In this step, the nodes containing the definition and implementation of the sequential micro-function to be executed in parallel are moved by the compiler from the algorithm code to a Reduce script, where the execution will actually take place.

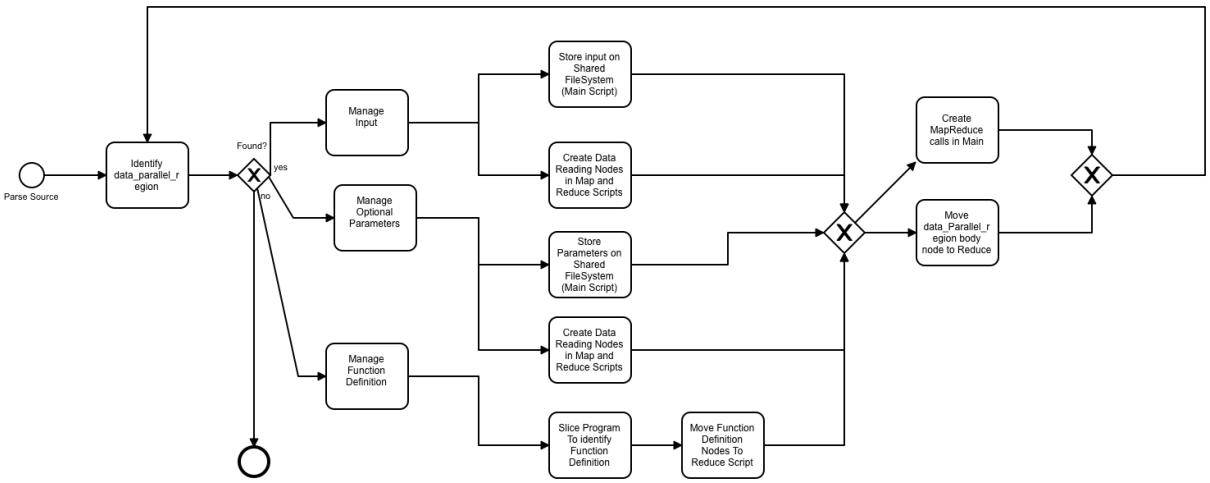


Figure 12.1: Description of the rules enforced by the compiler to transform a *data_parallel_region*. *Source: [BLM+18].*

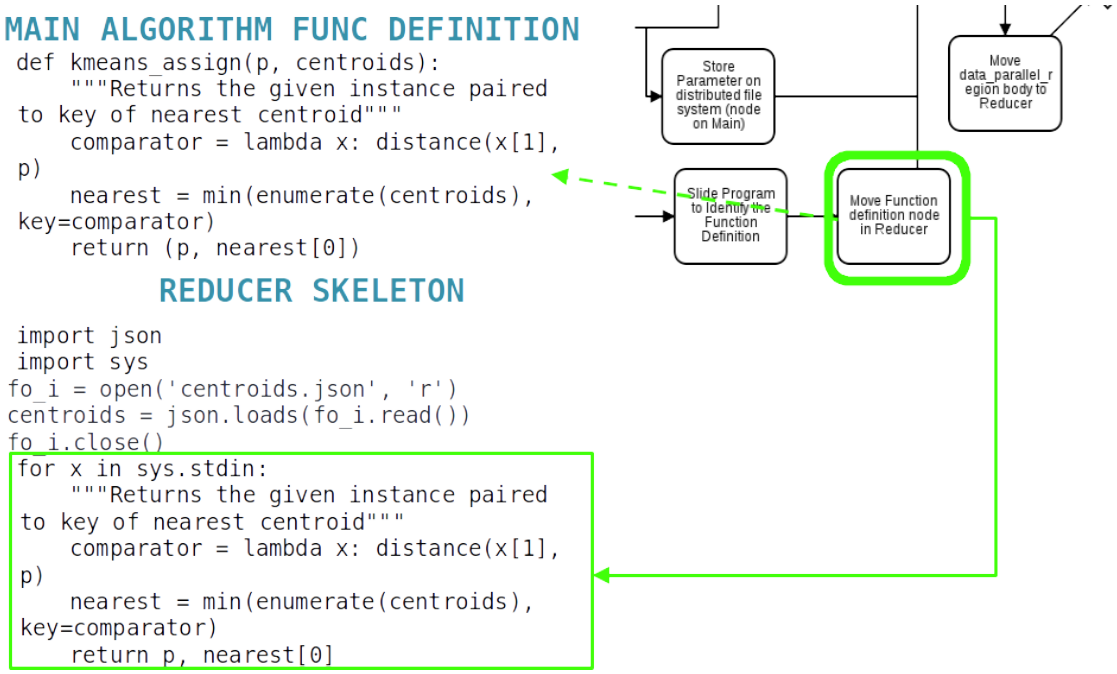


Figure 12.2: The node containing the micro-function definitions is migrated to a Reduce script. *Source: [BLM+18].*

12.5 Spark implementation of k-means based on parallel primitives

This section presents a complete example of the parallel primitives usage, and shows the entire process from writing the agnostic code based on the parallel primitives to its deployment on a target platform. Among the algorithms discussed in Section 12.3, k-means has been chosen as a usage example, dealing with a clustering problem.

K-means clustering has been applied to a custom data set. In particular, data have been generated by means of the *make_classification* function included in scikit-learn[1], a well-known Python library. In order to code the algorithm, the parallelizable operations have been identified, represented as functions and passed as parameters to the parallel primitives, as shown in Section 12.3.1. Python has been used as the programming language, due to its widespread employment for data analysis and due to its large support in terms of library packages. The algorithm code, based on the parallel primitives, has been transformed by the compiler, and then deployed on a Spark cluster.

As shown in Section 12.3.1, the parallelizable sections consist of the ASSIGN phase, which pairs the analyzed instance with the nearest centroid, and of the CENTROID phase, where the recomputation of centroids takes place. The first parallelizable section has been coded within the *kmeans_assign* function.

Listing 12.8: Implementation of the kmeans_assign function.

```
def kmeans_assign(p, centroids):
  comparator = lambda x: distance(x[1], p)
  nearest = min(centroids.items(), key=comparator)
  return {nearest[0]: (1, p)}
```

The outcome needs to be further processed by an aggregation function, named *kmeans_aggregate*.

Listing 12.9: Implementation of the kmeans_aggregate function.

```
def kmeans_aggregate(aggrs):
  totals = {}
  for aggr in aggrs:
    for (c, (nb, sb)) in aggr.items():
      if c in totals:
        (na, sa) = totals[c]
        totals[c]=(na+nb, map(sum, zip(sa, sb)))
      else:
        totals[c] = (nb, sb)
  return totals
```

Such functions have been passed as arguments to *data_parallel_region_with_reduction*, as *func* and *reduce_func* respectively.

[1] http://scikit-learn.org/

Listing 12.10: data_parallel_region_with_reduction call in the k-means code.

```
sums = data_parallel_region_with_reduction(data, kmeans_assign,
   kmeans_aggregate, centroids)
```

The second parallelizable section has been coded within the *kmeans_centroid* function.

Listing 12.11: Implementation of the kmeans_centroid function.

```
def kmeans_centroid(cluster_point, cluster_size):
  return cluster_point/cluster_size
```

This function has been used in conjunction with *data_parallel_region*.

Listing 12.12: data_parallel_region call in k-means code.

```
for x in sums.items():
  centroids[x[0]] = data_parallel_region(x[1][1], kmeans_centroid , x
     [1][0])
```

The compiler automatically transforms the primitives in parallel code for the target platform. In this example, k-means has been compiled in order to be executed on a Spark cluster. The following listing shows the compiled code for *data_parallel_region_with_reduction*, which calls *kmeans_assign* and *kmeans_aggregate*.

Listing 12.13: Compiled code for the data_parallel_region_with_reduction.

```
f = lambda j: kmeans_assign(j, centroids)
sums = mydata.map(f).reduce(kmeans_aggregate)
```

The *data_parallel_region_with_reduction* function call is compiled using the parallel computation structures provided by Spark. The *func* function is included in a lambda expression because of the presence of additional arguments, and then passed to the Spark *map* function. The need of a lambda expression is automatically detected by the compiler, and it is used to pass additional arguments as input to *func*. The *reduce_func* is directly passed as an argument to the Spark *reduce* function.

The following listing shows the compiled code for *data_parallel_region*, which calls the *kmeans_centroid* function.

Listing 12.14: Compiled code for the data_parallel_region.

```
for x in sums.items():
f = lambda j: kmeans_centroid(j, x[1][0])
centroids[x[0]] = x[1][1].map(f).collect()
```

Likewise *data_parallel_region_with_reduction*, the *data_parallel_region* function call is compiled using the parallel computation structures provided by Spark. Again, the *func* function is put in a lambda expression, and passed to the Spark *map* function. The result is finally extracted with a *collect* operation.

12.6 Experimental results

Soundness and performance checks have been made on the algorithm respectively to assure output consistency and to measure the differences in performance between the compiled distributed code and a manually written k-means implementation for the same target platform. The plain algorithm implementation for Spark has been written directly using some parallel Spark constructs, without the parallel primitives.

The soundness check aims to validate the output, which has to be consistent in both the manually written and the compiled algorithm. For this purpose, some test scripts have been created to choose the input data and to compare the outputs. Ordering operations have been applied to the output in order to enforce results comparability. The final ordered output has been saved to a file, which has then been compared using Linux $diff$ command. The performance check aims to measure the differences in performance between the manually written and the compiled algorithms. A collection of tests has been used to define measures to better understand the optimization level of the compiled code.

12.6.1 Dataset description

In order to perform experiments and to adequately assess whether the code written using the parallel primitives is sound and scalable, two kinds of datasets have been used: a freely available, widely known dataset and some generated data sets. The former has been used for the soundness check, since the performance check requires much larger data sets. Given that freely available large data sets are hard to find, some data have been generated in order to assess the algorithm's performance.

The dataset chosen to check the soundness of the compiled algorithm is the widely known *iris* dataset[2]. The generated sets share the *iris* dataset structure, but have a larger number of instances, features and classes. They have been generated using the *make_classification* function of the scikit-learn Python package. Data sets with increasing dimensions have been used, starting from 10,000 instances and growing to 100,000 and 1 million.

12.6.2 Performance check

The performance check aims to measure whether the compiled algorithm is as efficient as a manually optimized code. For this purpose, the elapsed time between the algorithms' start and termination has been measured and compared. The measure has been taken by using the *time* function of the *time* Python module.

The performance check has been performed on large generated data sets, to measure the scalability of the compiled algorithm. The tests have been conducted on data sets with 5 features and an increasing number of instances, starting from $10,000$ and growing to $100,000$ and 1 million. The number of slices used to parallelize the data is proportional to the data set cardinality

[2]https://archive.ics.uci.edu/ml/datasets/iris

(e.g. 50, 500 and 5,000 respectively). The dataset with 10,000 instances and divided into 50 slices is referred as the small dataset; analogously, the others are referred as the medium and the large respectively. The two versions of k-means have been tested on a cluster with an increasing number of worker nodes, starting from 2 and growing to 4, and 6. Each node has 4GB of RAM and 2 CPU.

Table 12.1 shows how the elapsed time of the compiled version of k-means scales with respect to the number of computational nodes and the number of instances used. The times shown in the table represent the average of the same experiment on 3 different runs.

# workers	Small	Medium	Large
2	47.22	312.52	2,881.83
4	22.65	202.52	1,779.22
6	21.06	165.29	1,333.02

Table 12.1: Scalability of the compiled version of k-means with respect to the number of workers.

Table 12.2 shows the comparison between the manual and the compiled k-means respectively. Again, the average elapsed time on 3 runs has been taken.

Algorithm	Small	Medium	Large
Manual k-means	22.89	163.18	1,315.04
Compiled k-means	21.06	165.29	1,333.02

Table 12.2: Elapsed time comparison between the manual and the compiled version of k-means.

The compiled version of k-means does not add a noticeable delay, compared to the manually written version. As can be seen in the experiment with 6 workers, which is the most affected by parallelization, the algorithms scale similarly by varying the dataset size.

12.7 Final remarks

This chapter has presented parallel programming primitives to define the parallelizable parts of algorithms in a uniform way, independent of existing analytic services. The rationale of this approach is to boost the development of big data analytics, fostering code reuse among different platforms and simplifying code optimization.

Programmers just have to identify parallel code fragments and encapsulate them into some parallel primitives, whereas parallellism as well as optimization are out of their duties. Parallelization is carried out by a compiler that is able to transform agnostically written code in a fully optimized, ready to be deployed application, tailored for a specific target platform. The compiler takes the algorithm definition as input, processes it, produces the necessary execution and deployment skeletons, choosing a suitable parallel computational pattern, and tuning the parameters of algorithms.

Part VI

Conclusion

13 Results achieved and future work

This chapter summarizes the most remarkable contributions of this thesis, and outlines ongoing and possible future work.

13.1 Methods for sentiment analysis

The first notable contribution of this thesis is the introduction of Markov chain techniques for sentiment analysis. In particular, our Markov chain introduced in Chapter 7 is able to perform both sentiment classification and transfer learning in cross-domain tasks. To the best of our knowledge, it is the first time that a Markov chain is used as a classifier for sentiment classification rather than for a support task as part-of-speech (POS) tagging.

Our Markov chain includes both terms and classes. The relationships between terms allow knowledge transfer from the source terms to the target terms, through the terms that are shared among domains. The relationships between terms and classes denote terms with a sentiment orientation that can be used for classification. Considering classes as Markov chain states is a novelty with reference to classification tasks, where the most common approach consists in building different Markov chains (e.g. one for each category) and evaluating the probability that a test document is being generated by each of them.

Also, we have proposed a variant of this basic approach that copes with sentences rather than considering each document as a whole (Section 7.3.1). A second variant has also been introduced so that the classification process is driven by polarity-bearing terms, which are able to discriminate among classes (Section 7.3.2).

Apart from the former variant, both the basic approach and the latter variant achieve accuracy comparable with the best methods in literature. Less parameters need to be tuned with respect to the state of the art, and less features are required to obtain such performance. These advantages make our Markov chain approach particularly appealing in big data scenarios, where scalability is as essential as effectiveness. Also, since our algorithm only relies on term co-occurrences, it can easily be applied to other languages.

The second remarkable contribution of this thesis is an investigation on deep learning for cross-domain sentiment classification. Paragraph vector, which is an unsupervised technique to learn distributed text representation that was not designed to perform knowledge transfer,

achieved comparable performance with our Markov chain in cross-domain tasks. The major outcome is that deep approaches to learn distributed text representations are able to extract domain-independent knowledge in an unsupervised fashion, so as to bridge the inter-domain semantic gap. This result suggests that a breakthrough of ad hoc cross-domain sentiment solutions can be obtained by combining distributed text representation and transfer learning techniques. Since paragraph vector can learn fixed-length feature representation from variable-length pieces of texts, and it is not affected by the curse of dimensionality, the breakthrough will also involve big data scenarios.

The suitability of distributed text representations as paragraph vector for transfer learning is confirmed by a really simple multi-source approach, where knowledge is extracted from N heterogeneous domains and the resulting model is applied to a different target domain. Using knowledge from multiple source domains is a naïve approach to enhance transfer learning; anyway, accuracy increases by about 2-3% on average independently of the dataset size. This outcome supports our belief that the breakthrough will also involve big data context, where very large data sets are required to be analyzed.

Memory-based deep neural networks as gated recurrent unit have been added to the investigation later on. With reference to cross-domain tasks, gated recurrent unit performs poorly with small-scale data (e.g. $2,000$ instances), achieves accuracy comparable with the other techniques with medium-scale data (e.g. $20,000$ instances), and even outperforms them with large-scale data (e.g. $100,000$ instances). The outcome suggests that gated recurrent unit needs many instances to learn bridging the inter-domain semantic gap. Once such instances are available, it is automatically able to align heterogeneous domains without explicit transfer learning mechanisms. This ability is supposedly due to GRU gates, which allow each unit working as a memory wherein relevant information can be stored and preserved through time.

Moreover, fine-tuning of a pre-trained model on a small sample of labeled target instances has been attempted to assess its impact on cross-domain as an explicit transfer learning mechanism. Paragraph vector does not take advantage of fine-tuning, since it is able to capture word semantics as well as word relationships without supervision. On the other hand, fine-tuning is beneficial to gated recurrent unit, because it acts as a transfer learning mechanism. The less training examples have originally been used to train the model on the source domain, the higher impact fine-tuning has on performance. As expected, greater amount of tuning data (e.g. 500 reviews rather than 250) brings to better performance with small-scale data. The impact of this factor decreases by augmenting the dataset cardinality, until eventually vanishing with large-scale data.

Strengthened by this first results, recent memory-based deep neural networks have been combined with word embeddings, a de facto standard in deep learning. Among the deep memory-based methods, gated recurrent unit and differentiable neural computer have been experimented. While in the former the memory mechanism is a part of the network structure, the latter is able to address and manage an external memory. Such an ability makes DNC one of the most innovative deep learning techniques, able to emulate reasoning and inference problems in natural language. Global vectors (GloVe) have been used in combination with both architectures for the initialization of their feature weights.

Experiments on the Amazon reviews corpus have shown that differentiable neural computer with GloVe dramatically outperforms state-of-the-art techniques for cross-domain sentiment classification. Transfer learning from a source domain to a target domain is supported by distributed word representations with small-scale datasets, and by memory mechanisms as the dataset size increases. Fine-tuning on a small sample of target instances is more useful to gated recurrent unit than differentiable neural computer, as the latter is less sensitive to noise, and few target samples are not enough to be relevant. Differentiable neural computer with GloVe feature weights achieves new state-of-the-art performance both in binary and fine-grained classifications on very large datasets built on the same Amazon reviews corpus. Finally, differentiable neural computer and gated recurrent unit achieve comparable performance with many techniques in single-sentence in-domain sentiment classification on the Stanford sentiment treebank. Small-scale training data and the absence of a mechanism to deal with sentence syntax are probably the reasons that prevent DNC from reaching the state-of-the-art performance.

13.2 Methods for stock market analysis

We have investigated whether the DJIA trend in a trading day is affected by the content of tweets posted in the previous days. In literature, some works have already tackled such a problem. However, in order to predict the stock market trend, they employed complex techniques to understand the semantic content of text documents. The aim of our work was to use a simple method, based on the well-known vector space model representation, and on a supervised classifier. We have also introduced a noise detection technique, both at the tweet level and at the instance level (i.e. aggregation of tweets), in order to filter out from data a large corpus of irrelevant tweets.

We have tested and compared the method on the same tweets dataset and DJIA trends in 2008 used by Bollen et al. [BMZ11]. Results have shown that even a simple classification model based on the vector space model achieves good accuracy, very close to 80%. Also, the noise detection technique introduced in Section 9.2.3 is able to identify the irrelevant tweets and instances, thus noise, in the training data. Our noise detection boosts accuracy, which achieves 88.9%, outperforming both our base classifier and the best prediction method based on social network posts illustrated in [BMZ11].

Moreover, we have added a trading protocol whose buy/sell operations are driven by the above mentioned prediction method. The test period used to compute the return on investment (ROI) is just three-month long. The ROI without and with the removal of noisy tweets are 56% and 84% respectively. Both results outperform previous DJIA trading approaches, such as stock market methods based only on the historical time series of DJIA prices [OM06], and recent advanced deep learning proposals [FM18, BYR17].

13.3 Recommendation methods for job search

This thesis has also introduced a novel algorithm for job recommendation, based on known co-occurrences between skills and potential job positions, which are exploited and elaborated through latent semantic analysis to discover hidden relationships. We have shown how the same data can be used to build a folksonomy of related job positions by means of a hierarchical clustering.

The methods have been tested using a set of public profiles extracted from LinkedIn, naturally subject to noise and inconsistencies; only some trivial pre-processing steps have been applied to denoise such data. Nonetheless, a hierarchical clustering has been produced where most groups actually include related job positions. Concerning recommendation, a quantitative experimental evaluation trivially based on real job positions has shown promising results. In half cases, the actual user occupation is suggested among the top 50 recommended positions out of more than $2,000$ possibilities. When the exact job is not hit, synonyms and similar occupations are generally suggested. Such a recommendation system can potentially aid both individuals seeking for occupations, whose abilities can effectively be endorsed, and recruiters, who have to evaluate the best candidates for specific jobs. Apart from the number of recommendations to be returned, the method has no parameter to be set. The absence of parameters to be tuned, and the large amount of data available on LinkedIn, make our method suitable for real world big data scenarios.

Another contribution of this thesis is the introduction of an MDP-based approach to recommend a personalized career pathway to a user, given her profile and career goal, and possibly other requirements. Career pathway recommendation, intended as the problem of finding the most likely sequence of jobs leading a user towards her career goal, is an emerging research thread that can be seen as a generalization of job recommendation. While a job recommender takes a snapshot of a user's profile and a job, and decides whether they match or not, a career pathway recommender knows the job the user is looking for, and iteratively recommends jobs in order to refine the user's profile until eventually the user is able to reach her goal.

A corpus of job postings has been used to mine relationships between jobs, and to build a job graph accordingly, where the likelihood to move across different clusters of jobs is defined. The job graph is unique, independent of users, and represents a network of jobs along with the relationships between them. Afterwards, the job graph has been used in combination with user-specific information, i.e. the user's profile and career goal, to build a Markov Decision Process (MDP) whose solution provide the user with a personalized pathway to follow in order to reach her career goal. Relying on the MDP theory, the recommended pathway is intrinsically the safest one, since every possible career evolution is considered.

The experiments showed that our framework is able to recommend a pathway that generally leads the user towards her career goal quicker than the user can do, provided that the job graph is generated with the right number of clusters. The contribution of the MDP-based recommender is particularly evident for long term career goals, where users are unlikely to find optimized pathways. We claim that our framework is general, and more complex formulas can easily be plugged in to take other factors into account, other than users' profiles.

13.4 Boosting the development of big data analytics

The last contribution of this thesis is the introduction of parallel primitives to implement big data mining algorithms in a vendor-agnostic fashion. Such primitives can simplify the implementation of algorithms, speeding up the development of big data analytics. Algorithms written by means of these primitives can then be compiled into skeletons, which are finally deployed on vendor platforms. The proposed approach supports code reuse, as code is written once and compiled by need for each target platform without additional effort. The compiler, introduced in [DMDE17], handles the transformation from a vendor-agnostic implementation to a vendor-specific code, managing and optimizing algorithm parallelization. Therefore, programmers' expertise in parallel programming is not required.

A practical example has been shown, emphasizing the advantages of this approach with respect to traditional vendor-specific implementations. The experimental evaluation has not only shown that such a transformation is feasible, but also that it produces optimized code, which can be executed as fast as manually written platform-specific code independently of the dataset size.

13.5 Ongoing and future work

13.5.1 Improvement of the Markov techniques for sentiment analysis

With reference to the Markov chain variant that deals with sentences (Section 7.3.1), a threshold can be introduced that establishes to what extent a sentence is likely to have the same polarity of the document wherein it appears. Another viable improvement consists in changing the way sentence labels are folded together to output the final category for the test document. Moreover, when a review is long, the final part usually bears the same sentiment of the entire text, because it contains a summary of the author's opinion. On the contrary when it is short, it is likely that the author immediately summarizes her opinion without using terms bearing conflicting sentiment. In both cases, taking only the last few sentences into account could be profitable. A further alternative is to use document splitting into sentences just to limit co-occurrences among terms rather than to affect the connections between terms and classes.

With respect to the Markov chain variant providing polarity-bearing state transitions (Section 7.3.2), there is no constraint that forces terms to redistribute probabilities to others having the same sentiment orientation. The inclusion of such a constraint may bring to a performance improvement.

In addition to the specific aspects proposed to enhance the performance of particular variants, further considerations need to be done. Transfer learning needs to be strengthened in order to improve the effectiveness in cross-domain tasks, for instance by increasing the number of steps in the Markov chain transition matrix, which might foster the classification phase. The algorithm could also suffer from the assumption to be in different states at the same time, made when a test document is required to be classified. In fact, since the step-by-step evolution of each state is independent of the others, context ends up to be overlooked. Ngrams, grammatical relations

among terms, or distributed word representations can be viable alternatives to take context into account. Other than enhancing algorithm effectiveness, performance in a 3-class setting (i.e. adding the neutral category) could also be tested. Finally, due to their generality, our Markov methods can be applied as is for text categorization by topic.

13.5.2 Enhancing deep learning approaches for sentiment analysis

Deep learning approaches have achieved outstanding performance in the identification of the sentiment orientation of a plain text. A possibility to improve transfer learning for cross-domain sentiment classification consists in combining the advantages of traditional machine learning techniques, as the Markov chain method introduced in Chapter 7, with distributed text representation methods, as paragraph vector and GloVe. On one hand, distributed text representations are able to learn word semantics without supervision; on the other hand, Markov chain provides a transfer learning mechanism to bridge the gap between the source and target domains for cross-domain sentiment classification.

Dynamic Memory Network (DMN) might also be added to the investigation, since it is provided with a memory mechanism, but it also performs well in single-sentence classification. It would be interesting to better understand the reason behind its behavior.

With reference to single-sentence classification, a mechanism to explicitly handle sentence syntax might be really useful in absence of a large amount of labeled training data. Finally, this study can be extended to cope with other emerging text classification problems where large datasets are completely unlabeled, such as for threads of conversational messages in social networks and discussion forums [DSM+16].

13.5.3 Expanding the investigation on stock market analysis

As future work on stock market analysis, we plan to further investigate possible correlations among different market indexes and stock options, expanding the analysis to other sources of unstructured text streams. In this context, the heterogeneity of multi-domain text sources may be taken into account and leveraged with novel cross-domain and transfer learning methods, such as [DMMP16, DMPS14b] and those presented in Chapter 7 and Chapter 8.

13.5.4 Improving the performance of job recommendation

One potential direction for further research on job recommendation is to devise a method that fits even better to a recruitment system, for example by swapping the roles of profiles and positions, so that a set of recommended candidates can be obtained for a given occupation. Another goal is to increase the accuracy of the job recommender introduced in Chapter 10. This goal can be fulfilled by testing other machine learning methods, as nearest neighbour classifiers, or even exploiting the generated hierarchy. Also, the vector representations of profiles, skills and positions could possibly be improved, for example by borrowing suitable weighting schemes

from text categorization [DMPS15]. Finally, we consider to test clustering and recommendation with more extended datasets, enlarging the number of profiles and including further information, such as a user's job history and education, and job description and requirements.

13.5.5 Enhancing the effectiveness of career pathway recommendation

We are currently performing further experiments on the approach introduced in Chapter 11. Mini-batch K-means can be compared with other clustering algorithms as well as with topic modeling techniques as Latent Dirichlet Allocation [BNJ03]. A simple heuristic can be used to recommend a career pathway in place for an MDP. Also, the number of representative skills of each cluster can be tuned.

Apart from these comparative evaluations, in order to enhance the effectiveness of the whole method, we plan to add further knowledge and constraints that can be exploited by the recommender to provide a more precise recommendation. Constraints may include salary, location, job duration, and so on. For instance, a user may want to reach her goal without performing jobs whose salary is lower than a threshold. Location could also affect the goodness of a job for a given user, and the likelihood for the user to get that job. Moreover, job duration can be exploited by the recommender to minimize the overall time required to reach the goal rather than the number of career hops.

The analysis can also be fostered by building a new corpus tailored to career pathway recommendation. Relevant information may include the user's actual career goal, whether or not she has been able to reach it, whether or not the reached goal actually meets her expectations, and further user-related as well as job-related constraints. Such knowledge paves the way for further investigation, starting from the framework and the baseline evaluation introduced in this thesis.

13.5.6 Supporting the development of big data analytics

Future work to accelerate the development of big data mining algorithms might include the addition of more parallel primitives, and the possibility to specify the particular computational pattern to be used in the target platform, such as divide and conquer, tree computation, map reduce, producer-consumer, bag of tasks, and so on.

Moreover, analytical as well as performance constraints may be introduced within the algorithm transformation process. Examples of analytical constraints are the maximum budget available to conduct the analysis, or the minimum number of worker nodes to be used. On the other hand, performance constraints may include the maximum time allowed for the analysis to be completed, the minimum tolerated accuracy, or further custom performance indicators.

Bibliography

[AAX13] Marta Arias, Argimiro Arratia, and Ramon Xuriguera. Forecasting with twitter data. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(1):8, 2013.

[ABB+11] Divyakant Agrawal, Philip Bernstein, Elisa Bertino, Susan Davidson, Umeshwas Dayal, Michael Franklin, Johannes Gehrke, Laura Haas, Alon Halevy, Jiawei Han, et al. Challenges and opportunities with big data 2011-1. 2011.

[ABE+14] Alexander Alexandrov, Rico Bergmann, Stephan Ewen, Johann-Christoph Freytag, Fabian Hueske, Arvid Heise, Odej Kao, Marcus Leich, Ulf Leser, Volker Markl, et al. The stratosphere platform for big data analytics. *The VLDB Journal—The International Journal on Very Large Data Bases*, 23(6):939–964, 2014.

[AF04] Werner Antweiler and Murray Z Frank. Is all that talk just noise? the information content of internet stock message boards. *The Journal of finance*, 59(3):1259–1294, 2004.

[AG05] Anthony Aue and Michael Gamon. Customizing sentiment classifiers to new domains: A case study. In *Proceedings of recent advances in natural language processing (RANLP)*, volume 1, pages 2–1. Citeseer, 2005.

[AH10] Sitaram Asur and Bernardo A Huberman. Predicting the future with social media. In *Proceedings of the 2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 492–499. IEEE Computer Society, 2010.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[Aiz64] Mark A Aizerman. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and remote control*, 25:821–837, 1964.

[Alp09] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[AOY12] Shaha T Al-Otaibi and Mourad Ykhlef. A survey of job recommender systems. *International Journal of Physical Sciences*, 7(29):5127–5142, 2012.

[AS+94] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[ASE14] Mohammed Amine Alimam, Hamid Seghiouer, and Yasyn Elyusufi. Building profiles based on ontology for career recommendation in e-learning context. In *multimedia computing and systems (ICMCS), 2014 international conference on*, pages 558–562. IEEE, 2014.

[AT05] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge & Data Engineering*, (6):734–749, 2005.

[ATK14] Nikolaos D Almalis, George A Tsihrintzis, and Nikolaos Karagiannis. A content based approach for recommending personnel for job positions. In *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*, pages 45–49. IEEE, 2014.

[AV09] George S Atsalakis and Kimon P Valavanis. Surveying stock market forecasting techniques–part ii: Soft computing methods. *Expert Systems with Applications*, 36(3):5932–5941, 2009.

[AVdV01] Jim Allen and Rolf Van der Velden. Educational mismatches versus skill mismatches: effects on wages, job satisfaction, and on-the-job search. *Oxford economic papers*, 53(3):434–452, 2001.

[AXV+11] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca Passonneau. Sentiment analysis of twitter data. In *Proceedings of the workshop on languages in social media*, pages 30–38. Association for Computational Linguistics, 2011.

[AZ12] Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.

[B+99] Rajkumar Buyya et al. High performance cluster computing: Architectures and systems (volume 1). *Prentice Hall, Upper SaddleRiver, NJ, USA*, 1:999, 1999.

[B+08] Dhruba Borthakur et al. Hdfs architecture guide. *Hadoop Apache Project*, 53:1–13, 2008.

[B+09] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[Bah10] Arash Bahrammirzaee. A comparative survey of artificial intelligence applications in finance: artificial neural networks, expert system and hybrid intelligent systems. *Neural Computing and Applications*, 19(8):1165–1195, 2010.

[BBL11] Ron Bekkerman, Mikhail Bilenko, and John Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.

[BDP07] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 440–447, 2007.

[Bel13] Richard Bellman. *Dynamic programming*. Courier Corporation, 2013.

[Bey11] M Beyer. Gartner says solving big data challenge involves more than just managing volumes of data. *Gartner*, 2011.

[BFRV12] Stephen D Brown, Robert J Francis, Jonathan Rose, and Zvonko G Vranesic. *Field-programmable gate arrays*, volume 180. Springer Science & Business Media, 2012.

[BG04] Igor A Bolshakov and Alexander Gelbukh. *Computational linguistics models, resources, applications*. Ciencia de la computación, 2004.

[BHBE10] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D Ernst. Haloop: efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment*, 3(1-2):285–296, 2010.

[BHMY99] L Douglas Baker, Thomas Hofmann, Andrew McCallum, and Yiming Yang. A hierarchical probabilistic model for novelty detection in text. *NIPS'99, Unpublished manuscript*, 1999.

[BHV+14] Mathieu Bastian, Matthew Hayes, William Vaughan, Sam Shah, Peter Skomoroch, Hyungjin Kim, Sal Uryasev, and Christopher Lloyd. Linkedin skills: large-scale topic extraction and inference. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 1–8. ACM, 2014.

[Bir12] Ewan Birney. The making of encode: lessons for big-data projects. *Nature*, 489(7414):49, 2012.

[BJC98] Tom Brotherton, Tom Johnson, and George Chadderdon. Classification and novelty detection using linear models and a class dependent-elliptical basis function neural network. In *Neural Networks Proceedings, 1998. IEEE World Congress on Computational Intelligence. The 1998 IEEE International Joint Conference on*, volume 2, pages 876–879. IEEE, 1998.

[BLM+18] Cesare Bandirali, Stefano Lodi, Gianluca Moro, Andrea Pagliarani, and Claudio Sartori. Parallel primitives for vendor-agnostic implementation of big data mining algorithms. In *2018 32nd International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 396–401, May 2018.

[blo11] *Drowning in numbers - digital data will flood the planet- and help us understand it better*. Economist Newspaper, 2011.

[BM92] Kirt C Butler and S Jamal Malaikah. Efficiency and inefficiency in thinly traded stock markets: Kuwait and saudi arabia. *Journal of Banking & Finance*, 16(1):197–210, 1992.

[BMBL09] Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. Network analysis in the social sciences. *science*, 323(5916):892–895, 2009.

[BMG16] Danushka Bollegala, Tingting Mu, and John Yannis Goulermas. Cross-domain sentiment classification using sentiment sensitive embeddings. *IEEE Transactions on Knowledge and Data Engineering*, 28(2):398–410, 2016.

[BMJM04] Patrick Buckley, Kathleen Minette, Dennis Joy, and Jeff Michaels. The use of an automated employment recruiting and screening system for temporary professional employees: A case study. *Human Resource Management: Published in Cooperation with the School of Business Administration, The University of Michigan and in alliance with the Society of Human Resources Management*, 43(2-3):233–241, 2004.

[BMMP13] Felipe Bravo-Marquez, Marcelo Mendoza, and Barbara Poblete. Combining strengths, emotions and polarities for boosting twitter sentiment analysis. In *Proceedings of the Second International Workshop on Issues of Sentiment Discovery and Opinion Mining*, page 2. ACM, 2013.

[BMP11] Johan Bollen, Huina Mao, and Alberto Pepe. Modeling public mood and emotion: Twitter sentiment and socio-economic phenomena. *Icwsm*, 11:450–453, 2011.

[BMZ11] Johan Bollen, Huina Mao, and Xiaojun Zeng. Twitter mood predicts the stock market. *Journal of computational science*, 2(1):1–8, 2011.

[BNJ03] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[BP98] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1):107 – 117, 1998.

[Bre96] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[BT03] Nicholas Barberis and Richard Thaler. A survey of behavioral finance. *Handbook of the Economics of Finance*, 1:1053–1128, 2003.

[Bue14] Ricardo Buettner. A framework for recommender systems in online social network recruiting: An interdisciplinary call to arms. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 1415–1424. IEEE, 2014.

[BWC13] Danushka Bollegala, David Weir, and John Carroll. Cross-domain sentiment classification using a sentiment sensitive thesaurus. *IEEE transactions on knowledge and data engineering*, 25(8):1719–1731, 2013.

[BYR17] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[Cad17] Carole Cadwalladr. The great british brexit robbery: how our democracy was hijacked. *The Guardian*, 20, 2017.

[Cat11] Rick Cattell. Scalable sql and nosql data stores. *Acm Sigmod Record*, 39(4):12–27, 2011.

[CBK09] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.

[CBVS14] Vikas Chirumamilla, Sruthi T Bhagya, Sasidhar Velpula, and Indira Sunkara. Novel approach to predict student placement chance with decision tree induction. *International journal of systems and technologies*, 2014.

[CBZ09] Edward Y Chang, Hongjie Bai, and Kaihua Zhu. Parallel algorithms for mining large-scale rich-media data. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 917–918. ACM, 2009.

[CCS11] Hao Chen, Yu Chen, and Douglas H Summerville. A survey on the application of fpgas for network infrastructure security. *IEEE Communications Surveys & Tutorials*, 13(4):541–561, 2011.

[CDN11] Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98, 2011.

[Cen10] Damon Centola. The spread of behavior in an online social network experiment. *science*, 329(5996):1194–1197, 2010.

[CGR+11] Michael D Conover, Bruno Gonçalves, Jacob Ratkiewicz, Alessandro Flammini, and Filippo Menczer. Predicting the political alignment of twitter users. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 192–199. IEEE, 2011.

[Chi99] Chia-Fen Chi. A study on job placement for handicapped workers using job analysis data. *International Journal of Industrial Ergonomics*, 24(3):337–351, 1999.

[CHL12] Alvin Chyan, Tim Hsieh, and Chris Lengerich. A stock-purchasing agent from sentiment analysis of twitter, 2012.

[CML14] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.

[CNB07] Guihong Cao, Jian-Yun Nie, and Jing Bai. Using markov chains to exploit word relationships in information retrieval. In *Large Scale Semantic Access to Content (Text, Image, Video, and Sound)*, pages 388–402. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE, 2007.

[CRF+11] Michael Conover, Jacob Ratkiewicz, Matthew R Francisco, Bruno Gonçalves, Filippo Menczer, and Alessandro Flammini. Political polarization on twitter. *Icwsm*, 133:89–96, 2011.

[CRS03] Soumen Chakrabarti, Shourya Roy, and Mahesh V Soundalgekar. Fast and accurate text classification via multiple linear discriminant projections. *The VLDB journal*, 12(2):170–185, 2003.

[Cuk10] Kenneth Cukier. *Data, data everywhere: A special report on managing information*. Economist Newspaper, 2010.

[CVMG+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[CW08] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[CXH+16] Tao Chen, Ruifeng Xu, Yulan He, Yunqing Xia, and Xuan Wang. Learning user and product distributed representations using a sequence model for sentiment analysis. *IEEE Computational Intelligence Magazine*, 11(3):34–44, 2016.

[DCG13] Luigi Di Caro and Matteo Grella. Sentiment analysis via dependency parsing. *Computer Standards & Interfaces*, 35(5):442–453, 2013.

[DDF+90] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.

[DFL+88] Susan T Dumais, George W Furnas, Thomas K Landauer, Scott Deerwester, and Richard Harshman. Using latent semantic analysis to improve access to textual information. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–285. Acm, 1988.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[DH00] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 71–80. ACM, 2000.

[DHZ+17] Mingjun Dai, Shansong Huang, Junpei Zhong, Chenguang Yang, and Shiwei Yang. Influence of noise on transfer learning in chinese sentiment classification using gru. In *2017 13th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pages 1844–1849. IEEE, 2017.

[DLP03] Kushal Dave, Steve Lawrence, and David M Pennock. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews. In *Proceedings of the 12th international conference on World Wide Web*, pages 519–528. ACM, 2003.

[DLY14] Zhi-Hong Deng, Kun-Hu Luo, and Hong-Liang Yu. A study of supervised term weighting scheme for sentiment analysis. *Expert Systems with Applications*, 41(7):3506–3513, 2014.

[DMB11] H Kristl Davison, Catherine Maraist, and Mark N Bing. Friend or foe? the promise and pitfalls of using social networking sites for hr decisions. *Journal of Business and Psychology*, 26(2):153–159, 2011.

[DMDE17] Beniamino Di Martino, Salvatore D'Angelo, and Antonio Esposito. A platform for mbdaaas based on patterns and skeletons: The python based algorithms compiler. In *Networking, Sensing and Control (ICNSC), 2017 IEEE 14th International Conference on*, pages 400–405. IEEE, 2017.

[DMMP16] Giacomo Domeniconi, Marco Masseroli, Gianluca Moro, and Pietro Pinoli. Cross-organism learning method to discover new gene functionalities. *Computer methods and programs in biomedicine*, 126:20–34, 2016.

[DMP+16] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, Karin Pasini, and Roberto Pasolini. Job recommendation from semantic similarity of linkedin users' skills. In *Proceedings of the 5th International Conference on Pattern Recognition Applications and Methods - Volume 1: ICPRAM,*, pages 270–277. INSTICC, SciTePress, 2016.

[DMPP] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. Cross-domain sentiment classification via polarity-driven state transitions in a markov model. In *Fred A., Dietz J., Aveiro D., Liu K., Filipe J. (eds) Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2015. Communications in Computer and Information Science, vol 631. Springer, Cham.*

[DMPP15] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. Markov chain based method for in-domain and cross-domain sentiment classification. In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR, (IC3K 2015)*, pages 127–137. INSTICC, SciTePress, 2015.

[DMPP17a] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. Learning to predict the stock market dow jones index detecting and mining relevant tweets. In *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR,*, pages 165–172. INSTICC, SciTePress, 2017.

[DMPP17b] Giacomo Domeniconi, Gianluca Moro, Andrea Pagliarani, and Roberto Pasolini. On deep learning in cross-domain sentiment classification. In *Proceedings of the 9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR,*, pages 50–60. INSTICC, SciTePress, 2017.

[DMPS14a] Giacomo Domeniconi, Gianluca Moro, Roberto Pasolini, and Claudio Sartori. Cross-domain text classification through iterative refining of target categories representations. In *KDIR*, pages 31–42, 2014.

[DMPS14b] Giacomo Domeniconi, Gianluca Moro, Roberto Pasolini, and Claudio Sartori. Iterative refining of category profiles for nearest centroid cross-domain text classification. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*, pages 50–67. Springer, 2014.

[DMPS15] Giacomo Domeniconi, Gianluca Moro, Roberto Pasolini, and Claudio Sartori. A study on term weighting for text categorization: a novel supervised variant of tf.idf. In *Proceedings of the 4th International Conference on Data Management Technologies and Applications*, 2015.

[DS07] Kaustav Das and Jeff Schneider. Detecting anomalous records in categorical datasets. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 220–229. ACM, 2007.

[DS12] Adnan Duric and Fei Song. Feature selection for sentiment analysis based on content and syntax models. *Decision support systems*, 53(4):704–711, 2012.

[dSG14] Cicero dos Santos and Maira Gatti. Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 69–78, 2014.

[DSM+16] Giacomo Domeniconi, Konstantinos Semertzidis, Gianluca Moro, Vanessa Lopez, Spyros Kotoulas, and Elizabeth M Daly. Identifying conversational message threads by integrating classification and data clustering. In *International Conference on Data Management Technologies and Applications*, pages 25–46. Springer, 2016.

[DXYY07] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Co-clustering based classification for out-of-domain documents. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 210–219. ACM, 2007.

[Efr94] Bradley Efron. Missing data, imputation, and the bootstrap. *Journal of the American Statistical Association*, 89(426):463–475, 1994.

[EIA11] Sudheep Elayidom, Sumam Mary Idikkula, and Joseph Alexander. A generalized data mining framework for placement chance prediction problems. *International Journal of Computer Applications (0975–8887) Volume*, 2011.

[ELW08] Andreas Eckhardt, Sven Laumer, and Tim Weitzel. Extending the architecture for a next-generation holistic e-recruiting system. In *CONF-IRM 2008 Proceedings*, page 27, 2008.

[EPF08] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. Mapreduce for data intensive scientific analyses. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 277–284. IEEE, 2008.

[Fam65] Eugene F Fama. The behavior of stock-market prices. *The journal of Business*, 38(1):34–105, 1965.

[FAM91] Eugene FAMA. Efficient capital markets: Ii the journal of finance. *Vol. XLVI*, (5):1575–1611, 1991.

[FAT+14] Adil Fahad, Najlaa Alshatri, Zahir Tari, Abdullah Alamri, Ibrahim Khalil, Albert Y Zomaya, Sebti Foufou, and Abdelaziz Bouras. A survey of clustering algorithms for big data: Taxonomy and empirical analysis. *IEEE transactions on emerging topics in computing*, 2(3):267–279, 2014.

[FC11] Teng-Kai Fan and Chia-Hui Chang. Blogger-centric contextual advertising. *Expert systems with applications*, 38(3):1777–1788, 2011.

[FHH+05] Eibe Frank, Mark Hall, Geoffre Holmes, Richard Kirkby, Bernhard Pfahringer, I Witten, Len Trigg, O Maimon Weka, and L Rokach. Data mining and knowledge discovery handbook. *US: Springer*, pages 1305–1314, 2005.

[Fle15] Lena Katharina Flecke. Utilizing facebook, linkedin and xing as assistance tools for recruiters in the selection of job candidates based on the person-job fit. B.S. thesis, University of Twente, 2015.

[FM18] Mirco Fabbri and Gianluca Moro. Dow jones trading with deep learning: The unreasonable effectiveness of recurrent neural networks. In *Proceedings of the 7th International Conference on Data Science, Technology and Applications - Volume 1: DATA,*, pages 142–153. INSTICC, SciTePress, 2018.

[FS97] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[FSCTR15] Marc Franco-Salvador, Fermín L Cruz, José A Troyano, and Paolo Rosso. Cross-domain polarity classification using a knowledge-enhanced meta-classifier. *Knowledge-Based Systems*, 86:46–56, 2015.

[FSV02] Paolo Frasconi, Giovanni Soda, and Alessandro Vullo. Hidden markov models for text categorization in multi-page documents. *Journal of Intelligent Information Systems*, 18(2-3):195–217, 2002.

[FYL05] Gabriel Pui Cheong Fung, Jeffrey Xu Yu, and Hongjun Lu. The predicting power of textual information on financial markets. *IEEE Intelligent Informatics Bulletin*, 5(1):1–10, 2005.

[GA12] Daniel Gayo-Avello. " i wanted to predict elections with twitter and all i got was this lousy paper"–a balanced survey on election prediction using twitter data. *arXiv preprint arXiv:1204.6441*, 2012.

[Gag17] Paul A Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, 2017.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[GBCB16] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[gdp18] General data protection regulation, 2018. Retrieved from: https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en, 2018-10-18.

[GE01] Gyozo Gidofalvi and Charles Elkan. Using news articles to predict stock price movements. *Department of Computer Science and Engineering, University of California, San Diego*, 2001.

[GG14] Anika Gupta and Deepak Garg. Applying data mining techniques in job recommender system for considering candidate job preferences. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on*, pages 1458–1465. IEEE, 2014.

[GGK+05] Daniel Gruhl, Ramanathan Guha, Ravi Kumar, Jasmine Novak, and Andrew Tomkins. The predictive power of online chatter. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 78–87. ACM, 2005.

[GK10] Eric Gilbert and Karrie Karahalios. Widespread worry and the stock market. In *ICWSM*, pages 59–65, 2010.

[GM04] Evgeniy Gabrilovich and Shaul Markovitch. Text categorization with many redundant features: using aggressive feature selection to make svms competitive with c4. 5. In *Proceedings of the twenty-first international conference on Machine learning*, page 41. ACM, 2004.

[GMP+09] Jeremy Ginsberg, Matthew H Mohebbi, Rajan S Patel, Lynnette Brammer, Mark S Smolinski, and Larry Brilliant. Detecting influenza epidemics using search engine query data. *Nature*, 457(7232):1012, 2009.

[GR11] John Gantz and David Reinsel. Extracting value from chaos. *IDC iview*, 1142(2011):1–12, 2011.

[GSBT05] Thomas L Griffiths, Mark Steyvers, David M Blei, and Joshua B Tenenbaum. Integrating topics and syntax. In *Advances in neural information processing systems*, pages 537–544, 2005.

[GSZ13] Manoochehr Ghiassi, James Skinner, and David Zimbra. Twitter brand sentiment analysis: A hybrid system using n-gram analysis and dynamic artificial neural network. *Expert Systems with applications*, 40(16):6266–6282, 2013.

[GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[GWR+16] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

[Had09] Apache Hadoop. Hadoop, 2009.

[HKM+88] John H Howard, Michael L Kazar, Sherri G Menees, David A Nichols, Mahadev Satyanarayanan, Robert N Sidebotham, and Michael J West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems (TOCS)*, 6(1):51–81, 1988.

[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*, pages 168–177. ACM, 2004.

[HLA11] Yulan He, Chenghua Lin, and Harith Alani. Automatically extracting polarity-bearing topics for cross-domain sentiment classification. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 123–131. Association for Computational Linguistics, 2011.

[HLN13] Michael Hagenau, Michael Liebmann, and Dirk Neumann. Automated news reading: Stock price prediction based on financial news using context-capturing features. *Decision Support Systems*, 55(3):685–697, 2013.

[Hoc98] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[How64] Ronald A Howard. Dynamic programming and markov processes. 1964.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[HZWS13] Wenxing Hong, Siting Zheng, Huan Wang, and Jianchao Shi. A job recommender system based on user clustering. *JCP*, 8(8):1960–1967, 2013.

[IC14] Ozan Irsoy and Claire Cardie. Modeling compositionality with multiplicative recurrent neural networks. *arXiv preprint arXiv:1412.6577*, 2014.

[JHS09] Wei Jin, Hung Hay Ho, and Rohini K Srihari. Opinionminer: a novel machine learning system for web opinion mining and extraction. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1195–1204. ACM, 2009.

[JO11] Yohan Jo and Alice H Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 815–824. ACM, 2011.

[Joa96] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996.

[Joh95] George H John. Robust decision trees: Removing outliers from databases. In *KDD*, pages 174–179, 1995.

[Jol11] Ian Jolliffe. Principal component analysis. In *International encyclopedia of statistical science*, pages 1094–1096. Springer, 2011.

[JW14] Saint John Walker. Big data: A revolution that will transform how we live, work, and think. 2014.

[Kas15] Kimberley Kasper. Half of all job seekers aren't in it for the long haul, jobvite job seeker nation study shows, 2015. Retrieved from: http://www.jobvite.com/press-releases/2015/half-job-seekers-arent-long-haul-jobvite-job-seeker-nation-study-shows/, 2018-10-18.

[KAYT90] Takashi Kimoto, Kazuo Asakawa, Morio Yoda, and Masakazu Takeoka. Stock market prediction system with modular neural networks. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pages 1–6. IEEE, 1990.

[KBQ14] Farhan Hassan Khan, Saba Bashir, and Usman Qamar. Tom: Twitter opinion mining framework using hybrid classification scheme. *Decision Support Systems*, 57:245–257, 2014.

[Kei07] Tobias Keim. Extending the applicability of recommender systems: A multilayer framework for matching human resources. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 169–169. IEEE, 2007.

[KGB14] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[KH04] Soo-Min Kim and Eduard Hovy. Determining the sentiment of opinions. In *Proceedings of the 20th international conference on Computational Linguistics*, page 1367. Association for Computational Linguistics, 2004.

[Kim14] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[KIO+16] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International Conference on Machine Learning*, pages 1378–1387, 2016.

[KM03] Dan Klein and Christopher D Manning. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics, 2003.

[KM11] Eamonn Keogh and Abdullah Mueen. Curse of dimensionality. In *Encyclopedia of machine learning*, pages 257–258. Springer, 2011.

[KN98] Edwin M Knox and Raymond T Ng. Algorithms for mining distance-based outliers in large datasets. In *Proceedings of the International Conference on Very Large Data Bases*. Citeseer, 1998.

[KR14] A Dinesh Kumar and Dr V Radhika. A survey on predicting student performance. *International Journal of Computer Science and Information Technologies*, 5(5):6147–6149, 2014.

[KS12] Akshi Kumar and Teeja Mary Sebastian. Sentiment analysis on twitter. *International Journal of Computer Science Issues (IJCSI)*, 9(4):372, 2012.

[Lam12] John Lamperti. *Stochastic processes: a survey of the mathematical theory*, volume 23. Springer Science & Business Media, 2012.

[Lan01] Doug Laney. 3d data management: Controlling data volume, velocity and variety. *META group research note*, 6(70):1, 2001.

[LAP99] Blake LeBaron, W Brian Arthur, and Richard Palmer. Time series properties of an artificial stock market. *Journal of Economic Dynamics and control*, 23(9):1487–1516, 1999.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[LD13] Fang Li and Tao Dong. Text categorization based on semantic cluster-hidden markov models. In *Advances in Swarm Intelligence*, pages 200–207. Springer, 2013.

[Lee07] In Lee. An architecture for a next-generation holistic e-recruiting system. *Communications of the ACM*, 50(7):81–85, 2007.

[Lee09] Ming-Chi Lee. Using support vector machine with a hybrid feature selection method to the stock trend prediction. *Expert Systems with Applications*, 36(8):10896–10904, 2009.

[LGZ+17] Huayu Li, Yong Ge, Hengshu Zhu, Hui Xiong, and Hongke Zhao. Prospecting the career development of talents: A survival analysis perspective. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 917–925. ACM, 2017.

[LHAY07] Yang Liu, Xiangji Huang, Aijun An, and Xiaohui Yu. Arsa: a sentiment-aware model for predicting sales performance using blogs. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 607–614. ACM, 2007.

[LHZ10] Fangtao Li, Minlie Huang, and Xiaoyan Zhu. Sentiment analysis with global topics and local dependency. In *AAAI*, volume 10, pages 1371–1376, 2010.

[Liu12] Bing Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

[LJ12] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[LJL12] Lianghao Li, Xiaoming Jin, and Mingsheng Long. Topic correlation analysis for cross-domain text classification. In *AAAI*, 2012.

[LKW15] Guy Lev, Benjamin Klein, and Lior Wolf. In defense of word embedding for generic text representation. In *International Conference on Applications of Natural Language to Information Systems*, pages 35–50. Springer, 2015.

[LLH12] Ming-Yen Lin, Pei-Yu Lee, and Sue-Chen Hsueh. Apriori-based frequent itemset mining algorithms on mapreduce. In *Proceedings of the 6th international conference on ubiquitous information management and communication*, page 76. ACM, 2012.

[LLKC11] Ming-Chih Lin, Anthony JT Lee, Rung-Tai Kao, and Kuo-Tay Chen. Stock price movement prediction using representative prototypes of financial reports. *ACM Transactions on Management Information Systems (TMIS)*, 2(3):19, 2011.

[Llo82] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[LM88] Andrew W Lo and A Craig MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *The review of financial studies*, 1(1):41–66, 1988.

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International Conference on Machine Learning*, pages 1188–1196, 2014.

[LMJ14] Carson Kai-Sang Leung, Richard Kyle MacKinnon, and Fan Jiang. Reducing the search space for big data mining for interesting patterns from uncertain data. In *big data (BigData congress), 2014 IEEE international congress on*, pages 315–322. IEEE, 2014.

[Loh12] Steve Lohr. The age of big data. *New York Times*, 11(2012), 2012.

[LR94] David D Lewis and Marc Ringuette. A comparison of two learning algorithms for text categorization. In *Third annual symposium on document analysis and information retrieval*, volume 33, pages 81–93, 1994.

[LSL+00] Victor Lavrenko, Matt Schmill, Dawn Lawrie, Paul Ogilvie, David Jensen, and James Allan. Mining of concurrent text and time series. In *KDD-2000 Workshop on Text Mining*, volume 2000, pages 37–44, 2000.

[LWD+11] Xiaodong Li, Chao Wang, Jiawei Dong, Feng Wang, Xiaotie Deng, and Shanfeng Zhu. Improving stock market prediction by integrating both market news and stock prices. In *International Conference on Database and Expert Systems Applications*, pages 279–293. Springer, 2011.

[LZ12] Bing Liu and Lei Zhang. A survey of opinion mining and sentiment analysis. In *Mining text data*, pages 415–463. Springer, 2012.

[MAC67] J MACQUEEN. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, 1967*, volume 1, pages 281–297. University of California Press, 1967.

[Mal96] Burton G Malkiel. A random walk down wall street: The best and latest investment advice money can buy. *Publisher: WW Norton & Company*, 1996.

[Mal03] Burton G Malkiel. The efficient market hypothesis and its critics. *Journal of economic perspectives*, 17(1):59–82, 2003.

[MBD+12] Andrew McAfee, Erik Brynjolfsson, Thomas H Davenport, DJ Patil, and Dominic Barton. Big data: the management revolution. *Harvard business review*, 90(10):60–68, 2012.

[MCB+11a] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela H Byers. Big data: The next frontier for innovation, competition, and productivity. 2011.

[MCB11b] Huina Mao, Scott Counts, and Johan Bollen. Predicting financial markets: Comparing survey, news, twitter and search engine data. *arXiv preprint arXiv:1112.1051*, 2011.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[MCMVULMR14] Eugenio Martínez-Cámara, M Teresa Martín-Valdivia, L Alfonso Urena-López, and A Rturo Montejo-Ráez. Sentiment analysis in twitter. *Natural Language Engineering*, 20(1):1–28, 2014.

[MDD09] Saif Mohammad, Cody Dunne, and Bonnie Dorr. Generating high-coverage semantic orientation lexicons from overtly marked words and a thesaurus. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2-Volume 2*, pages 599–608. Association for Computational Linguistics, 2009.

[MDR+06] Gilad Mishne, Maarten De Rijke, et al. Capturing global mood levels using blog posts. In *AAAI spring symposium: computational approaches to analyzing weblogs*, volume 6, pages 145–152, 2006.

[ME03] Hokey Min and Ahmed Emam. Developing the profiles of truck drivers for their successful recruitment and retention: a data mining approach. *International Journal of Physical Distribution & Logistics Management*, 33(2):149–162, 2003.

[Mei11] Erik Meijer. The world according to linq. *Communications of the ACM*, 54(10):45–51, 2011.

[MF11] Diana Maynard and Adam Funk. Automatic detection of political opinions in tweets. In *Extended Semantic Web Conference*, pages 88–99. Springer, 2011.

[MG+11] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[MG12] Anshul Mittal and Arpit Goel. Stock prediction using twitter sentiment analysis. *Standford University, CS229 (2011 http://cs229. stanford. edu/proj2011/GoelMittal-StockMarketPredictionUsingTwitterSentimentAnalysis. pdf)*, 15, 2012.

[MGL09] Prem Melville, Wojciech Gryc, and Richard D Lawrence. Sentiment analysis of blogs by combining lexical knowledge with text classification. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1275–1284. ACM, 2009.

[MHK14] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. Sentiment analysis algorithms and applications: A survey. *Ain Shams Engineering Journal*, 5(4):1093–1113, 2014.

[Mil95] George A Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[Mit04] M-A Mittermayer. Forecasting intraday stock price trends with text mining techniques. In *system sciences, 2004. proceedings of the 37th annual hawaii international conference on*, pages 10–pp. IEEE, 2004.

[MK06a] Marc-André Mittermayer and Gerhard Knolmayer. *Text mining systems for market response to news: A survey*. Institut für Wirtschaftsinformatik der Universität Bern, 2006.

[MK06b] Marc-Andre Mittermayer and Gerhard F Knolmayer. Newscats: A news categorization and trading system. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 1002–1007. Ieee, 2006.

[MKL+02] Dejan S Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. Peer-to-peer computing, 2002.

[MKWW06] Jochen Malinowski, Tobias Keim, Oliver Wendt, and Tim Weitzel. Matching people and jobs: A bilateral recommendation approach. In *null*, page 137c. IEEE, 2006.

[ML98] Bing Liu Wynne Hsu Yiming Ma and Bing Liu. Integrating classification and association rule mining. In *Proceedings of the fourth international conference on knowledge discovery and data mining*, 1998.

[MLS99] David RH Miller, Tim Leek, and Richard M Schwartz. A hidden markov model information retrieval system. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 214–221. ACM, 1999.

[MLW+07] Qiaozhu Mei, Xu Ling, Matthew Wondra, Hang Su, and ChengXiang Zhai. Topic sentiment mixture: modeling facets and opinions in weblogs. In *Proceedings of the 16th international conference on World Wide Web*, pages 171–180. ACM, 2007.

[MM08] David Mimno and Andrew McCallum. Modeling career path trajectories, 2008.

[MNS05] Stephen Marsland, Ulrich Nehmzow, and Jonathan Shapiro. On-line novelty detection for autonomous mobile robots. *Robotics and Autonomous Systems*, 51(2):191–206, 2005.

[MP12] Hye-Jin Min and Jong C Park. Identifying helpful reviews based on customer's mentions about experiences. *Expert Systems with Applications*, 39(15):11830–11838, 2012.

[MPD+] Gianluca Moro, Roberto Pasolini, Giacomo Domeniconi, Andrea Pagliarani, and Andrea Roli. Prediction and trading of dow jones from twitter: A boosting text mining method with relevant tweets identification. In *Fred A., Dietz J., Aveiro D., Liu K., Filipe J. (eds) Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2017. Communications in Computer and Information Science*. To appear.

[MPPS18] Gianluca Moro, Andrea Pagliarani, Roberto Pasolini, and Claudio Sartori. Cross-domain & in-domain sentiment analysis with memory-based deep neural networks. In *10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management - Volume 1: KDIR,*, 2018. To appear.

[MRCZ12] Alejandro Moreo, M Romero, JL Castro, and Jose Manuel Zurita. Lexicon-based comments-oriented news sentiment analyzer system. *Expert Systems with Applications*, 39(10):9166–9180, 2012.

[MS94] Elke Mittendorf and Peter Schäuble. Document and passage retrieval based on hidden markov models. In *SIGIR'94*, pages 318–327. Springer, 1994.

[MS03] Markos Markou and Sameer Singh. Novelty detection: a review—part 1: statistical approaches. *Signal processing*, 83(12):2481–2497, 2003.

[MS11] Yelena Mejova and Padmini Srinivasan. Exploring feature definition and selection for sentiment classifiers. In *ICWSM*, 2011.

[MSC+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*, pages 3111–3119, 2013.

[MV12] Isa Maks and Piek Vossen. A lexicon model for deep sentiment analysis and opinion mining applications. *Decision Support Systems*, 53(4):680–688, 2012.

[MVN13] Rodrigo Moraes, JoãO Francisco Valiati, and Wilson P GaviãO Neto. Document-level sentiment classification: An empirical comparison between svm and ann. *Expert Systems with Applications*, 40(2):621–633, 2013.

[MWH94] Gina J Medsker, Larry J Williams, and Patricia J Holahan. A review of current practices for evaluating causal models in organizational behavior and human resources management research. *Journal of management*, 20(2):439–464, 1994.

[MWK08] Jochen Malinowski, Tim Weitzel, and Tobias Keim. Decision support for team staffing: An automated relational recommendation approach. *Decision Support Systems*, 45(3):429–447, 2008.

[MWW13] Yuexin Mao, Wei Wei, and Bing Wang. Twitter volume spikes: analysis and application in stock trading. In *Proceedings of the 7th Workshop on Social Network Mining and Analysis*, page 4. ACM, 2013.

[MWWL12] Yuexin Mao, Wei Wei, Bing Wang, and Benyuan Liu. Correlating s&p 500 stocks with twitter data. In *Proceedings of the first ACM international workshop on hot topics on interdisciplinary social networks research*, pages 69–72. ACM, 2012.

[Nat08] Nature. Big data. *https://www.nature.com/collections/wwymlhxvfs*, 2008.

[ND10] John Nickolls and William J Dally. The gpu computing era. *IEEE micro*, 30(2), 2010.

[NGL97] Hwee Tou Ng, Wei Boon Goh, and Kok Leong Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *ACM SIGIR Forum*, volume 31, pages 67–73. ACM, 1997.

[Nog11a] Yuki Noguchi. Following digital breadcrumbs to big data gold. *National Public Radio*, 3:56–88, 2011.

[Nog11b] Yuki Noguchi. The search for analysts to make sense of big data. *National Public Radio*, 2011.

[NY03] Tetsuya Nasukawa and Jeonghee Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77. ACM, 2003.

[OCA13] Nuno Oliveira, Paulo Cortez, and Nelson Areal. Some experiments on modeling stock market behavior using investor sentiment analysis and posting volume from twitter. In *Proceedings of the 3rd International Conference on Web Intelligence, Mining and Semantics*, page 31. ACM, 2013.

[OHL+08] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[OM06] Niall O'Connor and Michael G Madden. A neural network approach to predicting stock exchange movements using external factors. In *Applications and Innovations in Intelligent Systems XIII*, pages 64–77. Springer, 2006.

[ORS+08] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM, 2008.

[PAB+] Andrea Pagliarani, Oznur Alkan, Adi Botea, Gianluca Moro, and Elizabeth Daly. A skillset-based approach to modelling job transitions and recommending career pathways. To be submitted to journal Expert Systems with Applications.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[PCG11] Ioannis Paparrizos, B Barla Cambazoglu, and Aristides Gionis. Machine learned job recommendation. In *Proceedings of the fifth ACM Conference on Recommender Systems*, pages 325–328. ACM, 2011.

[PL+08] Bo Pang, Lillian Lee, et al. Opinion mining and sentiment analysis. *Foundations and Trends® in Information Retrieval*, 2(1–2):1–135, 2008.

[PLL12] Yi-Cheng Pan, Hung-Yi Lee, and Lin-Shan Lee. Interactive spoken document retrieval with suggested key terms ranked by a markov decision process. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(2):632–645, 2012.

[PLM+04] Joël Plisson, Nada Lavrac, Dr Mladenić, et al. A rule based approach to word lemmatization. 2004.

[PMPD] Andrea Pagliarani, Gianluca Moro, Roberto Pasolini, and Giacomo Domeniconi. Transfer learning in sentiment classification with deep neural networks. In *Fred A., Dietz J., Aveiro D., Liu K., Filipe J. (eds) Knowledge Discovery, Knowledge Engineering and Knowledge Management. IC3K 2017. Communications in Computer and Information Science*. To appear.

[PNS+10] Sinno Jialin Pan, Xiaochuan Ni, Jian-Tao Sun, Qiang Yang, and Zheng Chen. Cross-domain sentiment classification via spectral feature alignment. In *Proceedings of the 19th international conference on World wide web*, pages 751–760. ACM, 2010.

[Por80] Martin F Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[PP10] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREc*, volume 10, pages 1320–1326, 2010.

[PR12] Indranil Palit and Chandan K Reddy. Scalable and parallel boosting with mapreduce. *IEEE Transactions on Knowledge and Data Engineering*, 24(10):1904–1916, 2012.

[PRS13] Alexander Porshnev, Ilya Redkin, and Alexey Shevchenko. Improving prediction of stock market indices by analyzing the psychological states of twitter users. 2013.

[PSM14] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[PT10] Georgios Paltoglou and Mike Thelwall. A study of information retrieval weighting schemes for sentiment analysis. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 1386–1395. Association for Computational Linguistics, 2010.

[Put14] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[PY+10] Sinno Jialin Pan, Qiang Yang, et al. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[Qiu93] Liwen Qiu. Markov models of search state patterns in a hypertext information retrieval system. *Journal of the American Society for Information Science*, 44(7):413–427, 1993.

[QR07] Bo Qian and Khaled Rasheed. Stock market prediction with multiple classifiers. *Applied Intelligence*, 26(1):25–33, 2007.

[Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Qui93] John Ross Quinlan. *C4. 5: Programs for Machine Learning*, volume 1. Morgan Kaufmann, 1993.

[QZHZ09] Likun Qiu, Weishi Zhang, Changjian Hu, and Kai Zhao. Selc: a self-supervised model for sentiment classification. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 929–936. ACM, 2009.

[QZX+18] Chuan Qin, Hengshu Zhu, Tong Xu, Chen Zhu, Liang Jiang, Enhong Chen, and Hui Xiong. Enhancing person-job fit for talent recruitment: An ability-aware neural network approach. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 25–34. ACM, 2018.

[R+11] Philip Russom et al. Big data analytics. *TDWI best practices report, fourth quarter*, 19(4):1–34, 2011.

[RBB02] Robert S Rubin, William H Bommer, and Timothy T Baldwin. Using extracurricular activity as an indicator of interpersonal skill: Prudent evaluation or recruiting malpractice? *Human Resource Management: Published in Cooperation with the School of Business Administration, The University of Michigan and in alliance with the Society of Human Resources Management*, 41(4):441–454, 2002.

[RBS00] Rachael Rafter, Keith Bradley, and Barry Smyth. Personalised retrieval for online recruitment services. In *The BCS/IRSG 22nd Annual Colloquium on Information Retrieval (IRSG 2000), Cambridge, UK, 5-7 April, 2000*, 2000.

[RHC+12] Eduardo J Ruiz, Vagelis Hristidis, Carlos Castillo, Aristides Gionis, and Alejandro Jaimes. Correlating financial time series with micro-blogging activity. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 513–522. ACM, 2012.

[RHN+14] Tajul Rosli Razak, Muhamad Arif Hashim, Noorfaizalfarid Mohd Noor, Iman Hazwam Abd Halim, and Nur Fatin Farihin Shamsul. Career path recommendation system for uitm perlis students using fuzzy logic. In *Intelligent and Advanced Systems (ICIAS), 2014 5th International Conference on*, pages 1–5. IEEE, 2014.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[RML14] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[Rok12] Lior Rokach. Introduction to machine learning, 2012. Retrieved from: https://www.slideshare.net/liorrokach/introduction-to-machine-learning-13809045, 2018-10-23.

[RR12] Antonio Reyes and Paolo Rosso. Making objective decisions from subjective data: Detecting irony in customer reviews. *Decision Support Systems*, 53(4):754–760, 2012.

[RRS00] Sridhar Ramaswamy, Rajeev Rastogi, and Kyuseok Shim. Efficient algorithms for mining outliers from large data sets. *SIGMOD Rec.*, 29(2):427–438, May 2000.

[RRS15] Francesco Ricci, Lior Rokach, and Bracha Shapira. Recommender systems: introduction and challenges. In *Recommender systems handbook*, pages 1–34. Springer, 2015.

[RS99] Miguel E Ruiz and Padmini Srinivasan. Hierarchical neural networks for text categorization. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 281–282. ACM, 1999.

[RS10] Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[RS14] Tushar Rao and Saket Srivastava. Twitter sentiment analysis: How to hedge your bets in the stock markets. In *State of the Art Applications of Social Network Analysis*, pages 227–247. Springer, 2014.

[Sar00] Ramesh R Sarukkai. Link prediction and path analysis using markov chains. *Computer Networks*, 33(1):377–386, 2000.

[SC06] Robert Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using financial news articles. *AMCIS 2006 Proceedings*, page 185, 2006.

[SC09] Robert P Schumaker and Hsinchun Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Transactions on Information Systems (TOIS)*, 27(2):12, 2009.

[Sch15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[Sci11] Science. Special online collection: dealing with big data. *http://www.sciencemag.org/site/special/data/*, 2011.

[Ser09] Richard Serfozo. *Basics of applied stochastic processes*. Springer Science & Business Media, 2009.

[SF11] Carlos N Silla and Alex A Freitas. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery*, 22(1-2):31–72, 2011.

[SHB05] Guy Shani, David Heckerman, and Ronen I Brafman. An mdp-based recommender system. *Journal of Machine Learning Research*, 6(Sep):1265–1295, 2005.

[SHMN12] Richard Socher, Brody Huval, Christopher D Manning, and Andrew Y Ng. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1201–1211. Association for Computational Linguistics, 2012.

[SHX02] Qing Song, Wenjie Hu, and Wenfang Xie. Robust support vector machine with bullet hole image classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 32(4):440–448, 2002.

[SMGE11] Abhinav Shrivastava, Tomasz Malisiewicz, Abhinav Gupta, and Alexei A Efros. Data-driven visual similarity for cross-domain image matching. *ACM Transactions on Graphics (ToG)*, 30(6):154, 2011.

[SPH+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*, pages 151–161. Association for Computational Linguistics, 2011.

[SPW+13] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[SR90] David B. Skalak and Edwina L. Rissland. Inductive learning in a mixed paradigm setting. In *Proceedings of the Eighth National Conference on Artificial Intelligence - Volume 2*, AAAI'90, pages 840–847. AAAI Press, 1990.

[SR11] RM Samant and S Rao. The effect of noise in automatic text classification. In *Proceedings of the International Conference & Workshop on Emerging Trends in Technology*, pages 557–558. ACM, 2011.

[SR15] Dilpreet Singh and Chandan K Reddy. A survey on platforms for big data analytics. *Journal of Big Data*, 2(1):8, 2015.

[STSW14] Timm O Sprenger, Andranik Tumasjan, Philipp G Sandner, and Isabell M Welpe. Tweets and trades: The information content of stock microblogs. *European Financial Management*, 20(5):926–957, 2014.

[SW05] Ralf Steinmetz and Klaus Wehrle. *Peer-to-peer systems and applications*, volume 3485. Springer, 2005.

[SWNF12] Zheng Siting, Hong Wenxing, Zhang Ning, and Yang Fan. Job recommender systems: a survey. In *Computer Science & Education (ICCSE), 2012 7th International Conference on*, pages 920–924. IEEE, 2012.

[TBP11] Mike Thelwall, Kevan Buckley, and Georgios Paltoglou. Sentiment in twitter events. *Journal of the American Society for Information Science and Technology*, 62(2):406–418, 2011.

[TBT+11] Maite Taboada, Julian Brooke, Milan Tofiloski, Kimberly Voll, and Manfred Stede. Lexicon-based methods for sentiment analysis. *Computational linguistics*, 37(2):267–307, 2011.

[TBW08] Lori Foster Thompson, Phillip W Braddy, and Karl L Wuensch. E-recruitment and the benefits of organizational web appeal. *Computers in Human Behavior*, 24(5):2384–2398, 2008.

[TD99] David MJ Tax and Robert PW Duin. Support vector domain description. *Pattern recognition letters*, 20(11):1191–1199, 1999.

[Tet07] Paul C Tetlock. Giving content to investor sentiment: The role of media in the stock market. *The Journal of Finance*, 62(3):1139–1168, 2007.

[TLCV15] Chun-Wei Tsai, Chin-Feng Lai, Han-Chieh Chao, and Athanasios V Vasilakos. Big data analytics: a survey. *Journal of Big Data*, 2(1):21, 2015.

[TM08] Donald Thomas and Philip Moorby. *The Verilog® Hardware Description Language*. Springer Science & Business Media, 2008.

[TP12] Mikalai Tsytsarau and Themis Palpanas. Survey on mining subjective data on the web. *Data Mining and Knowledge Discovery*, 24(3):478–514, 2012.

[TQL15] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, pages 1422–1432, 2015.

[TS00] James D Thomas and Katia Sycara. Integrating genetic algorithms and text learning for financial prediction. *Data Mining with Evolutionary Algorithms*, pages 72–75, 2000.

[TSJ+09] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff, and Raghotham Murthy. Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629, 2009.

[TSM15] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015.

[Tur02] Peter D Turney. Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 417–424. Association for Computational Linguistics, 2002.

[TvdS13] Cem Tekin and Mihaela van der Schaar. Distributed online big data classification using context information. In *Communication, Control, and Computing (Allerton), 2013 51st Annual Allerton Conference on*, pages 1435–1442. IEEE, 2013.

[TWC08] Songbo Tan, Yuefen Wang, and Xueqi Cheng. Combining learn-based and lexicon-based techniques for sentiment detection without using labeled examples. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 743–744. ACM, 2008.

[VC12] G Vinodhini and RM Chandrasekaran. Sentiment analysis and opinion mining: a survey. *International Journal*, 2(6):282–292, 2012.

[VID] Adrián Seara Vieira, Eva Lorenzo Iglesias, and Lourdes Borrajo Diz. Study and application of hidden markov models in scientific text classification.

[VLL+10] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec):3371–3408, 2010.

[VMD+13] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 5. ACM, 2013.

[WCL+98] Beat Wuthrich, Vincent Cho, Steven Leung, D Permunetilleke, K Sankaran, and Jian Zhang. Daily stock market forecast from textual web data. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, volume 3, pages 2720–2725. IEEE, 1998.

[WFHP16] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[WG14] Haibing Wu and Xiaodong Gu. Reducing over-weighting in supervised term weighting for sentiment analysis. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 1322–1330, 2014.

[WHF07] Kangning Wei, Jinghua Huang, and Shaohong Fu. A survey of e-commerce recommender systems. In *Service systems and service management, 2007 international conference on*, pages 1–5. IEEE, 2007.

[WHS+05] Theresa Wilson, Paul Hoffmann, Swapna Somasundaran, Jason Kessler, Janyce Wiebe, Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Opinionfinder: A system for subjectivity analysis. In *Proceedings of hlt/emnlp on interactive demonstrations*, pages 34–35. Association for Computational Linguistics, 2005.

[WJL16] Xingyou Wang, Weijie Jiang, and Zhiyong Luo. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2428–2437, 2016.

[WWH05] Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005.

[WYD+13] Xindong Wu, Kui Yu, Wei Ding, Hao Wang, and Xingquan Zhu. Online feature selection with streaming features. *IEEE transactions on pattern analysis and machine intelligence*, 35(5):1178–1192, 2013.

[WZ03] Xindong Wu and Shichao Zhang. Synthesizing high-frequency rules from different data sources. *IEEE Transactions on Knowledge and Data Engineering*, 15(2):353–367, 2003.

[WZ08] Xindong Wu and Xingquan Zhu. Mining with noise knowledge: error-aware data mining. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 38(4):917–932, 2008.

[WZWD14] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107, 2014.

[XDYY08] Gui-Rong Xue, Wenyuan Dai, Qiang Yang, and Yong Yu. Topic-bridged plsa for cross-domain text classification. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 627–634. ACM, 2008.

[XLGC12] Huiqi Xu, Zhen Li, Shumin Guo, and Keke Chen. Cloudvista: interactive and economical visual cluster analysis for big data in the cloud. *Proceedings of the VLDB Endowment*, 5(12):1886–1889, 2012.

[XPC12] Tao Xu, Qinke Peng, and Yinzhao Cheng. Identifying the semantic orientation of terms using s-hal for sentiment analysis. *Knowledge-Based Systems*, 35:279–289, 2012.

[XSH+06] Rong Xu, Kaustubh Supekar, Yang Huang, Amar Das, and Alan Garber. Combining text classification and hidden markov modeling techniques for structuring randomized clinical trial abstracts. In *AMIA Annual Symposium Proceedings*, volume 2006, page 824. American Medical Informatics Association, 2006.

[XW00] Jinxi Xu and Ralph Weischedel. Cross-lingual information retrieval using hidden markov models. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13*, pages 95–103. Association for Computational Linguistics, 2000.

[XYX+15] Huang Xu, Zhiwen Yu, Hui Xiong, Bin Guo, and Hengshu Zhu. Learning career mobility and human activity patterns for job change analysis. In *2015 IEEE International Conference on Data Mining (ICDM)*, pages 1057–1062. IEEE, 2015.

[XYY+16] Huang Xu, Zhiwen Yu, Jingyuan Yang, Hui Xiong, and Hengshu Zhu. Talent circle detection in job transition networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 655–664. ACM, 2016.

[XZZ+17] Tong Xu, Hengshu Zhu, Chen Zhu, Pan Li, and Hui Xiong. Measuring the popularity of job skills in recruitment market: A multi-criteria approach. *arXiv preprint arXiv:1712.03087*, 2017.

[YAC07] Xing Yi, James Allan, and W Bruce Croft. Matching resumes and jobs based on relevance models. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 809–810. ACM, 2007.

[Yan95] Yiming Yang. Noise reduction in a statistical approach to text categorization. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263. ACM, 1995.

[YB08] Kwan Yi and Jamshid Beheshti. A hidden markov model-based text classification of medical documents. *Journal of Information Science*, 2008.

[YB13] Kwan Yi and Jamshid Beheshti. A text categorization model based on hidden markov models. In *Proceedings of the Annual Conference of CAIS/Actes du congrès annuel de l'ACSI*, 2013.

[YWCC13] Liang-Chih Yu, Jheng-Long Wu, Pei-Chann Chang, and Hsuan-Shou Chu. Using a contextual entropy model to expand emotion words and their intensity for the sentiment classification of stock market news. *Knowledge-Based Systems*, 41:89–97, 2013.

[ZCF+10] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[ZE+11] Paul Zikopoulos, Chris Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.

[ZESD14] Julie Zide, Ben Elman, and Comila Shahani-Denning. Linkedin and recruitment: How profiles differ across occupations. *Employee Relations*, 36(5):583–604, 2014.

[ZFG11] Xue Zhang, Hauke Fuehres, and Peter A Gloor. Predicting stock market indicators through twitter "i hope it is not as bad as i fear". *Procedia-Social and Behavioral Sciences*, 26:55–62, 2011.

[ZHH07] Yuzheng Zhai, Arthur Hsu, and Saman K Halgamuge. Combining news and technical indicators in daily stock price trends prediction. In *International symposium on neural networks*, pages 1087–1096. Springer, 2007.

[ZHL+15] Yuhong Zhang, Xuegang Hu, Peipei Li, Lei Li, and Xindong Wu. Cross-domain sentiment classification-feature divergence, polarity divergence or both? *Pattern recognition letters*, 65:44–50, 2015.

[ZJZ10] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[ZL15a] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[ZL15b] Xiang Zhang and Yann LeCun. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

[ZYN14] Yingya Zhang, Cheng Yang, and Zhixiang Niu. A research of job recommendation system based on collaborative filtering. In *Computational Intelligence and Design (ISCID), 2014 Seventh International Symposium on*, volume 1, pages 533–538. IEEE, 2014.

[ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015.

[ZZX+16] Chen Zhu, Hengshu Zhu, Hui Xiong, Pengliang Ding, and Fang Xie. Recruitment market trend analysis with sequential latent variable models. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 383–392. ACM, 2016.

List of Figures

List of Tables