Alma Mater Studiorum – Università di Bologna

DOTTORATO DI RICERCA IN

Ingegneria Elettronica, Informatica e delle Telecomunicazioni

Ciclo XXVII

Settore concorsuale di afferenza: 09/H1
Settore scientifico disciplinare: ING-INF/05

LEARNING METHODS AND ALGORITHMS FOR SEMANTIC TEXT CLASSIFICATION ACROSS MULTIPLE DOMAINS

Presentata da: Roberto Pasolini

Coordinatore Dottorato
Prof. Alessandro Vanelli-Coralli

Relatore
Prof. Gianluca Moro

Correlatore
Prof. Claudio Sartori

Esame finale anno 2015

Abstract

Information is nowadays a key resource: machine learning and data mining techniques have been developed to extract high-level information from great amounts of data. As most data comes in form of unstructured text in natural languages, research on text mining is currently very active and dealing with practical problems.

Among these, text categorization deals with the automatic organization of large quantities of documents in priorly defined taxonomies of topic categories, possibly arranged in large hierarchies. In commonly proposed machine learning approaches, classifiers are automatically trained from pre-labeled documents: they can perform very accurate classification, but often require a consistent training set and notable computational effort.

Methods for cross-domain text categorization have been proposed, allowing to leverage a set of labeled documents of one domain to classify those of another one. Most methods use advanced statistical techniques, usually involving tuning of parameters. A first contribution presented here is a method based on nearest centroid classification, where profiles of categories are generated from the known domain and then iteratively adapted to the unknown one. Despite being conceptually simple and having easily tuned parameters, this method achieves state-of-the-art accuracy in most benchmark datasets with fast running times.

A second, deeper contribution involves the design of a domain-independent model to distinguish the degree and type of relatedness between arbitrary documents and topics, inferred from the different types of semantic relationships between respective representative words, identified by specific search algorithms. The application of this model is tested on both flat and hierarchical text categorization, where it potentially allows the efficient addition of new categories during classification. Results show that classification accuracy still requires improvements, but models generated from one domain are shown to be effectively able to be reused in a different one.

To everyone who somehow supported me

Contents

Introduction

Nowadays, across many contexts, information of all kinds is produced at fast rates. Many things can be considered "information": the current temperature in a room, the details of a purchase in a store, a post on a social network and so on. From information, especially in large quantities, useful knowledge can be extracted: the details of many sales of a store may give useful indications about products often purchased together, while a collection of public messages about some product or service can reveal what people generally think about it. However, the extraction of useful knowledge from very large amounts of data is not always trivial, especially to be performed manually.

Data mining research is dedicated to developing processes for automated extraction of useful, high-level information hidden within large amounts of data. It has many applications in business, finance, science, society and so on. Data mining is mostly based on machine learning, the study of algorithms to analyze a set of raw data and extract from it a knowledge model which encapsulates recurring patterns within it and allows to make more or less accurate predictions on future data. Machine learning algorithms can solve various task, such as classification of items or events according to prominent characteristics, such as classifying reviews of a product as positive or negative.

Ordinary machine learning techniques work on structured data, where every piece of information is well distinguishable and computers can easily handle it. On the other way, unstructured data exist, which are easily interpreted by humans in small amounts, but not directly understandable by computers. A prominent part of such data is constituted by free text in

natural language, such as English or Italian, in units of different lengths, ranging from short messages on Twitter to publicly available whole books. Such text data, whose examples have been given above, can contain like other very valuable information, but automatic analysis is necessary to efficiently extract such knowledge in a usable form.

Text mining is a branch of data mining studying techniques for automatic treatment of free text. This is generally achieved by processing text to represent it with structured forms, which can then be analyzed by usual learning algorithms. A very common form to efficiently represent every document of a large collection is the bag of words, a multiset of relevant words or other related features contained in the text.

In this context, a quite common task is text categorization, consisting in the automatic organization of a possibly very large set of documents into distinct, meaningful categories. Categorization is usually carried out according to separate different topics discussed in document, but the general description involves also tasks like spam filtering in e-mail and identification of positive and negative reviews of a product. Organization of documents by topics can be useful in contexts like dividing books in sections, separating news articles by categories and organizing Web pages in hierarchical directories.

Most methods for automatic text categorization are based on machine learning: given a training set of documents already labeled with respective correct categories, they are reduced to bags of words and, using suitable learning algorithms, a knowledge model is inferred from them, which is able to classify further documents within the same categories. This approach automates the learning process, but requires a consistent training set, which may not be priorly available and thus require considerable effort to be built.

To overcome this problem, at least in some circumstances, methods for cross-domain classification have been proposed, where a knowledge model to classify a set of documents within a target domain is obtained by leveraging a set of pre-labeled documents of a source domain which is allowed to have some differences from the former. Most methods to perform this task are based on advanced statistical techniques, which transform source data to adapt them to the target or convert data of both domains to a common representation.

Here is proposed a simple method for cross-domain text categorization based on explicit document-like representations of categories, likely to some existing methods for standard classification, which are created from the

source domain and then iteratively refined to adapt them to the target domain. From experimental evaluations, this approach appears to be effective in finding good representations of categories of target domain and consequently in accurately classifying documents within them.

Keeping valid the idea of explicit category profiles, it is subsequently introduced an improved method to compare them to documents, based on the analysis of semantic relationships held between respective relevant words: it is theorized that these can suggest whether and how documents and categories are related, according to implicit rules which are more or less globally valid across any domain. This leads to the construction of a domain-independent model of semantic knowledge which encapsulates this knowledge. To support the analysis of semantic relationships between words, a couple of alternative algorithms have been developed to identify non-obvious relationships from sequences of primitive links between concepts provided by the WordNet lexical database.

The use of this general knowledge model is tested on text categorization. The proposal is to use a set of pre-labeled documents to extract a knowledge model which encapsulates these rules, then to use this model to classify incoming documents by comparing them to representation of relevant categories. Under the assumption of its general validity across multiple domains, a model extracted from a training set should be fairly able to support classification of documents in an arbitrarily different domain, for which is only necessary to have bag-of-words-like representations of categories which can be extracted very efficiently from pre-labeled documents.

This model potentially allows to build a classification system where categories can be added and removed at any time just by providing suitable representations which are very efficiently generated from example documents: this is possible as the knowledge model itself is decoupled from specific words and categories of its training data. Some basic experiments are performed to evaluate the accuracy of classification with this approach and to investigate to which degree the knowledge model can be applied to a domain different from the one of its training documents.

Contributions

Here are summarized the main contributions presented in the thesis. In the sub-points, references to related publications and works currently under

review are given.

- Starting from general data mining and machine learning, a discussion of text mining tasks and techniques is given, with focus on text categorization and machine learning techniques usually employed within it. This also serves to describe techniques which are employed in the subsequently presented work.

- Cross-domain text categorization, where documents of a target domain are classified by leveraging knowledge of a slightly different source domain, is discussed. A novel method based on nearest centroid classification is presented: the source domain is used to extract initial category profiles, which are then iteratively refined according to most similar documents in the target domain, until a stable configuration is reached. Compared to the state of the art, the method yields better or comparable results with a more conceptually simple algorithm, which can be implemented easily, exposes few parameters to be tuned and runs fast. A couple of variants are separately discussed to further improve robustness with respect to parameters and running times.

  - G. Domeniconi, G. Moro, R. Pasolini, C. Sartori. Cross-domain text categorization through iterative refining of target categories representations. *6th International Conference on Knowledge Discovery and Information Retrieval (KDIR)*, 2014. (awarded as Best Student Paper)

  - (undergoing review process) G. Domeniconi, G. Moro, R. Pasolini, C. Sartori. Iterative refining of category profiles for nearest centroid cross-domain text classification. Submitted to *Knowledge Discovery, Knowledge Engineering and Knowledge Management* (*Communications in Computer and Information Science* series, Springer).

- Finally, is introduced the idea of a domain-independent model predicting whether and how documents and category profiles are related from the analysis of semantic relationships held between respective representative words. Text categorization is presented as a concrete application of this model, indicating how it is built and then used.

A couple of non-trivial algorithms are proposed to search indirect semantic relationships from the primitive links given by the WordNet database. Specific methods are proposed for both flat and hierarchical classification, along with results of experiments to evaluate the classification accuracy and the generality of the model.

  - G. Domeniconi, G. Moro, R. Pasolini, C. Sartori. Domain-independent text categorization. *2nd Italian Workshop on Machine Learning and Data Mining (MLDM.it)* at the *XIII Conference of the Italian Association for Artificial Intelligence*, 2013.
  - (undergoing review process) G. Domeniconi, G. Moro, R. Pasolini, C. Sartori. Towards topic-independent text classification: a novel semantic learning method for hierarchical corpora. Submitted to *Journal of Machine Learning Research* (Microtome Publishing).

- As an additional work outside of the text mining research, is described a novel method for identification of malicious traffic in a network based on distributed data clustering. In the proposed approach, different network nodes gather aggregated statistics about incoming traffic from standard SNMP agents, on which cluster analysis is then run in a distributed fashion: nodes exchange summary information of respective clusters with neighbors, in order to have a shared clustering model, so that each node has knowledge of classes of traffic even if not detected by itself.

  - (undergoing review process after major revision) W. Cerroni, G. Moro, R. Pasolini, M. Ramilli. Decentralized detection of network attacks through P2P data clustering of SNMP data. Submitted to *Computers and Security* (Elsevier).

Structure of the thesis

Chapter 1 gives a general introduction of automated text analysis along with its motivations, describing most prominent examples of high and low level tasks usually tackled. In Chapter 2, the most recurring techniques employed in text mining are described in detail. Chapter 3 focuses on text categorization, showing some specific techniques and methods presented

in literature. Chapter 4 goes in detail on cross-domain text categorization, presenting known approaches and exposing the novel method based on iterative refining of category profiles. Chapter 5 introduces the domain-independent knowledge model and its application to text categorization, including discussion of search of semantic relationships between words and presentation of flat and hierarchical variants along with experiments. Finally, Chapter 6 briefly sums up the work and indicates potential directions for further research.

In the end, outside of the text mining scope, Appendix A presents the work dealing with network security based on distributed data mining.

Conventions

Throughout the thesis, cited single *words* (or equivalently *terms*) which may appear in text documents are typeset in sans-serif font. Names of *categories* of documents, introduced in Chapter 3 and generally denoting topics, are instead distinguished by SMALL CAPITALS.

Technical remarks

All the experimental evaluations which are part the work presented in the thesis have been performed on virtualized hardware, with the use of up to 6 parallel processing cores.

Tests have been run within a dedicated software framework implemented in Java. The following third-party tools and libraries have been extensively used:

- the *WEKA* software suite for machine learning [44] (`http://www.cs.waikato.ac.nz/ml/weka/`),
- the *Java WordNet Interface* (JWI) [37] (`http://projects.csail.mit.edu/jwi/`).

Where cited, version 3.1 of the WordNet database (described in §2.6.1) has been used.

Chapter 1

Data and Text Mining

Starting from data mining in general, a general overview about automatic handling of text in natural language is given in this chapter, including the motivations for this research and the general tasks which are usually tackled.

1.1 Data mining

Information is a key resource throughout many contexts. A part of scientific research is collection and analysis of data from experiments. Many areas of engineering are similarly based on analyzing available data to find optimal solutions to problems. In the business context, information can be gathered and used to drive strategic decisions and actions. These are only some general examples, many other exist.

A possible problem in this vision is the *lack* of information, generally meaning that possessed data is not enough to have a sufficiently accurate knowledge of the phenomenon under analysis.

Nonetheless, with the continuous development of processing, storage and communication means, a situation of *excess* of information can be easily reached. While a limited quantity of information can be feasibly analyzed by human workers, difficulties can be met in extracting useful information from an overly abundant quantity of data.

At this end, the fields of *data mining* and *knowledge discovery* are generally focused on the development of methods and techniques for the non-trivial extraction of high-level information from potentially enormous

quantities of *raw* data [39]. Data mining is successfully applied in business, research and many fields and is used to perform tasks such as discovering patterns in data and partitioning elements of large sets into smaller, meaningful groups according to similar characteristics.

1.1.1 Applications

Data mining techniques in general have a wide range of applications. Here only some examples are given.

In the business context, a great amount of data is or can be gathered from operations of a company. Taking a chain of stores as an example, data of all purchases from all stores can be gathered, associated where possible to specific customers. This data, suitably stored in *data warehouses*, can usually be accessed through on-line analytical processing (OLAP) tools to have condensed summary information about different aspects of the state of the business, such as sales for each product category or in each store.

Through data mining, it is additionally possible to perform automatic analysis of this data in order to highlight non-obvious patterns within them. A typical task is *market basket analysis*, where past purchases data is analyzed in order to locate items which are often sold together; another useful application is *customers segmentation*, where distinct groups of identifiable customers with similar habits are tried to be delineated. All this high-level information can be useful for example when taking marketing decisions regarding discounts and promotional offers. Further examples in business and financial contexts are analysis of applications for credits by financial institutions or evaluation of risks and detection of frauds by insurance companies.

Research is also a prominent area where data mining techniques are employed. In general, it can be used to discover recurring patterns in the possibly large quantities of data obtained by observations and experiments in various fields. In genetics, specific techniques for *sequence mining* are used to discover correlations between DNA sequences and phenomenons like susceptibility to some diseases.

Another field of application of data mining are sensor networks, used in some environments to monitor conditions such as temperature or air pollution in other to gather data about them with space and time references. Given the possibly large extension of such networks and often the limited communication capabilities, they are the natural context for distributed data mining methods, cited below.

Data mining is primarily intended to analyze structured data, but techniques for analysis of data such as images and music (other than text) have been developed.

1.1.2 Machine learning

While a data mining or knowledge discovery process generally involves different steps, ranging from the collection and processing of data to be analyzed to the consumption of final results, the key step is obviously the conversion of the great and not easily understandable amount of "raw" data into compact, high-level information which can be easily interpreted.

This task is usually based on *machine learning* techniques, which are generally used to infer *knowledge models* from data of heterogeneous form and nature: these models can highlight interesting features of the analyzed data and can be used to make predictions regarding data which arrives in the future.

In general, input data is composed of *observations* or *instances*, each corresponding to one item or event of a set to be analyzed: these can be for example transactions (purchases) of a store. Observations are distinguishable from each other by a set of *features*, indicating which are the properties of each observation. Seeing this data as a table of a relational database, observations correspond to records (rows), while features correspond to columns.

To be used as input for machine learning algorithms, available data generally needs to undergo some pre-processing steps to obtain a set of observations with convenient features. This can involve joining data from multiple tables of a relational database, in order to obtain in practice a single table where all information is condensed with no or minimal loss. This can lead to a consistent number of features, which can make the analysis cumbersome (*dimensionality problem*): specific methods exist to perform selection of these features and discard those least useful in the analysis. Learning methods are generally able to work only with specific types of data: some of them in their basic versions can only deal with discrete values, such as integer numbers or categorical data (elements from a finite set), while others are specialized on continuous ranges of values, which can anyway be discretized in order to be treated by other algorithms.

Different machine learning algorithms exist, differing mainly for the type of knowledge which is extracted. A rather common need is to extract knowl-

edge from known items or past events in order to somehow assign labels to them or to new items or events according to some specific important characteristics of them. This generic description can be applied to two different tasks.

- ***Classification*** is the task of labeling new items with a set of predefined set of *classes*, which represent categories in which is convenient to subdivide such items. In general, this process involves building a *training* set of observation which are already correctly labeled with the same classes; a machine learning algorithm then analyzes data to find relevant correlations between features and class labels and extracts from them a *classifier*, which is ideally able to correctly label any observation represented with the same features. As data provided for training must be priorly labeled, algorithms of this type are said to be *supervised*. An example of applications of this general task is insurance fraud detection (indemnity requests are classified as either suspect or not).

- ***Clustering*** is the task of partitioning data in some groups, said *clusters*, about which no prior knowledge is given. The goal of a clustering algorithm is, given a set of data, to identify clusters within this set so that each contains observations which are very similar to each other and significantly different from those of other clusters; possibly, also new observations can then be organized in the same clusters. Differently from classification, clustering algorithms are *unsupervised*, as they analyze data with no priorly assigned group labels. Among concrete applications described above, customers segmentation is an example of clustering problem.

For each of these two general tasks, as well as for others, different approaches can be used: which are the most convenient ones usually depends from the nature of the data under analysis. For example, in Section 3.5, some general techniques for classification will be shown, along with considerations on how much they are suitable to text categorization.

1.2 Text mining

Data mining has been introduced as the extraction of information from large amounts of data. Such data can come in different forms, each requiring specific techniques to be handled. Machine learning algorithms, as discussed above, are suitable to handle *structured* data, where single pieces of information are organized according to some model and each of them is easily distinguishable from the others, as usually happens for example in a relational database.

A type of data which in general can't be handled directly by these algorithms is free *text* written in natural language, such as English or Italian. While a computer can check for equality between two elements from a finite set and can operate with numbers, it can not trivially perform tasks on portions of text which are relatively simple for persons. Examples of such tasks include recognizing the topic treated in the text or the opinion of the writer about it and making a summary of the contents.

Text mining (or *text analysis*) is the research field which deals with extracting useful, high-level information from text: it can be seen in practice as data mining applied on natural language text.

Most text mining-related literature has its roots in data mining, machine learning and related fields as well as in *information retrieval*, which generally refers to automatic retrieval from a vast data collection of information relevant to some need. Most research on this field is focused on retrieval from text collections and proposed many of the techniques used in text mining.

Likely to data mining, a text mining process generally involves a preliminary pre-processing phase, which in this case is needed to translate raw text into a structured form, upon which statistical analysis tools such as machine learning algorithms can be applied. The typical choice is to represent documents as vectors, according to a variety of techniques spawned from information retrieval research, which can then be processed with many known methods. Coverage on these techniques will be given in Chapter 2.

1.2.1 Applications

Text mining techniques can be exploited in different contexts where abundance of textual data is produced.

A prominent source of such data is the World Wide Web, especially referring to the so-called "Web 2.0", where each user can publish self-generated content: this includes sites such as discussion forums, blogs with readers comments and social networks. Due to the large diffusion and usage of these platforms, the amount of data produced every day by their users is huge, and a substantial part of it consists of textual data.

Texts produced and published by Web users include posts on forums and social networks, product reviews on e-commerce services and reviews of services (such as restaurants and hotels) and creative works (such as movies and video games) on specialized sites. Given its nature, this data can potentially indicate the general opinion of people about a variety of topics, persons and things, making it an highly valuable asset. For example, the automated analysis of a bunch of reviews of a particular product or service may reveal the most relevant strengths and weaknesses experienced by its users.

However, the huge amount of such data requires specialized methods to extract the needed high-level information. Text mining techniques are very useful in this context, enabling the selection of data relevant to some topic and the summarization of its content. A relatively recent and currently very active branch of text mining research is *sentiment analysis*, which is specialized in understanding the positive or negative polarity of a portion of text expressing an opinion, which may be referred to anything (a person, a brand, a political party, etc.).

Another necessity regarding massive collection of text is its automatic organization. This necessity arises for collections of text documents which are ought to be consulted by users, such as books in a library or stories in a news site: these texts are generally grouped into categories, consistently with topics treated by them. When the number of documents is even moderately high, manually grouping them can be a cumbersome task, so many methods have been devised to automatize this process.

Other than browsing documents by predefined categories, a user may want to retrieve documents which are the most relevant to a specific *query* decided by him or her, as commonly happens when using a Web search engine. This is a problem tackled since earlier research, in the specific branch of *information retrieval*, which will be discussed later in Section 2.1.

1.3 High-level text mining tasks

Above, a general overview of the possible applications of text mining methods has been given. In the following, a shallow distinction is given of the specific problems which are tackled by text mining, along with a quick overview of related literature. These tasks are here regarded as "high-level" as they can be applied to large collections of documents as a whole.

1.3.1 Text categorization

Text categorization (or *classification*) generally refers to the organization of a set of text documents into categories of a priorly defined set, often representing topics of discussion such as "science", "arts", "sports" and so on. This general description may refer to many practical problems in a wide range of complexity, such as classifying mail messages as useful or spam, labeling articles of a news site with meaningful topics and indexing web pages and sites in hierarchical directories.

In the vast majority of recent works, automatic text classification is tackled through the use of machine learning algorithms, which use training documents as input to infer classifiers able to coherently label subsequent documents. This approach requires to pre-process documents to represent them as vectors, after defining a suitable set of predictive features [107].

Text classification will be treated in detail throughout the rest of the thesis, starting from Chapter 3 (although with references to general techniques presented in Chapter 2).

1.3.2 Sentiment analysis

Sentiment analysis (or *opinion mining*) refers to the extraction from a collection of text of the general attitude expressed regarding some particular object [91, 74]. This branch of text analysis includes some different cases.

In a conceptually simple case, sentiment analysis means classifying a set of text documents (e.g. reviews of a movie) as expressing a *positive* or *negative* (or maybe *neutral*) opinion. To perform a finer distinction, documents may be labeled with a rating, implying different possible levels of positive or negative polarity, as expressed for example with ratings "from one to five stars". To extract even more useful information, the analysis may identify different aspects of the same object and evaluate the average

polarity for each of them (e.g. reviews of a movie may declare that it has an *interesting* plot, but that it is *poorly* directed). Another possibility is to understand whether a polarity is present at all, distinguishing subjective from objective texts.

Sentiment analysis methods can be in some cases considered as specific applications of other known task: classifying reviews as either positive or negative can be seen for example as a text categorization problem. Anyway, specific techniques are often employed when dealing with opinions, for example the extraction of composite phrases rather than single words and the use of external knowledge indicating the polarity of each of them.

1.3.3 Text clustering

Text clustering refers to the identification of groups (*clusters*) of similar documents in a potentially large set of them [76, 1]. Likely to text classification described above, the goal is to organize documents in groups; the difference is that the groups are not priorly given.

As in general data clustering, groups should be created so that each document is largely similar to others of the same group and different from those of other groups. Also the used techniques are equivalent or similar to known ones, in particular hierarchical and partitive (e.g. *k-means* or variants) clustering.

Clustering results may be used to automatically organize a vast collection in groups of similar documents in order to ease their browsing. In this case, to be more useful in practice, especially if their number is large, clusters should be automatically labeled with some meaningful names or descriptions of the documents in each. This task is a form of text summarization, further discussed below. The organization of documents in clusters may even be used as an indexing method to support other tasks, such as retrieval of documents relevant to an arbitrary query.

1.3.4 Document summarization

Document summarization generally refers to obtaining a summary of contents from a document or a set thereof, in form of keyphrases, keywords or sentences [26].

A very diffused approach is summarization by *extraction* consists into giving as summary a set of words, phrases or sentences appearing in the

text, picking the ones deemed to be more representative of the contents [43]. Many works select such items by means of machine learning or using other probabilistic models, such as hidden Markov models.

Another solution is summarization by *abstraction*, where new sentences which describe the contents of the text are generated. This potentially allows for more concise and fluently readable summaries, but constitutes a much harder challenge, due to the necessity to generate non-trivial portions of general language. The extraction of information needed for the summary also requires advanced techniques to understand natural language.

Other than summarization itself, a non-trivial problem discussed in literature is *evaluation* of summarization methods. While for tasks like text categorization the experimental evaluation is simply based on verifying that predictions on a test set match a *gold standard* and is trivially automated, this method is not easily appliable to summarization. Summaries for a same document (or set thereof) given by different human expert are very likely to differ from each other and is therefore difficult to define a unique correct solution to be used as gold standard [77]; manual evaluation of automated summaries can be a solution, but requires considerable effort.

1.4 Natural language processing

Natural language processing (NLP) generally refers to methods and techniques allowing a computer to directly handle text in human language. While this definition also includes tasks like automatic generation of such text, the focus here is on reading and analysis of natural language: the general goal is to infer the underlying structure of read text at different possible levels of detail.

The distinction made here between techniques described in this section and text mining tasks discussed above is mainly based on the scope: while tasks like text classification are inherently carried out on whole collections of documents to extract high-level information, techniques ascribable to NLP are generally employed to perform a more fine-grained analysis of text and many of them can be applied on single documents independently from the others. Citing [55], the difference is that text mining is "the discovery and extraction of interesting, non-trivial knowledge from free or unstructured text", while NLP is "the attempt to extract a fuller meaning representation from free text".

According to the specific needs, NLP techniques can be useful by themselves to perform some specific tasks or can be leveraged to aid in addressing the needs presented in the previous sections.

In the following, some common NLP tasks are presented.

1.4.1 Part-Of-Speech Tagging

Each word in a text belongs to one *part of speech* (POS), a grammatical category denoting its function in a sentence. Most common parts of speech are *nouns* and *verbs*, usually present in every phrase, while examples of other parts are *articles*, *pronouns* and *conjunctions*.

While for a human is often trivial to distinguish among such different parts of speech (it is usually learned at school), there are some concerns for computers. Given the potential ambiguity of natural language, the correct part of speech for a word might not be identifiable by itself and may depend in a complex way from its context. For example, the word `set` might be a noun ("the *set* of natural numbers"), a verb ("parameters must be *set*") or an adjective ("the table is *set* for two people").

Specialized algorithms for part-of-speech tagging (POS tagging) exist, analyzing sentences in their entirety to determine the correct POS for each of their words. A POS tagging algorithm must have knowledge of the grammatical structures of the specific language to be analyzed: this may be given in form of predefined rules, but is common to use a training set of preliminary tagged text from which such knowledge is automatically extracted, for example in form of hidden Markov models [23].

Known examples of these tagged datasets, which are also used for evaluating correctness of the methods, are the Brown Corpus and the Penn Treebank Dataset. Each of such datasets is based on a specific *tags set*, which defines the possible labels which can be attributed to words. While commonly only about ten grammatical categories are considered (nouns, verbs, etc.), tags sets commonly contain many more elements in order to make finer distinctions within these categories: for example, in both the two datasets cited above proper nouns are distinguished from the rest and also singular and plural nouns are separated.

1.4.2 Word Sense Disambiguation

POS tagging, identifying the components of the underlying grammatical structure of text, is a first step in extracting its meaning. Anyway, a subsequent step is to identify the specific concepts expressed by each word. Again, the difficulty of this task lies in the ambiguity of the language, where many *homonimies* exist: a word may refer to more than one meaning. A typical example is the word bank, which as a noun may refer to either a financial institution (as in "a *bank* account") or a portion of land ("the *bank* of the river"). Similarly to how distinguishing the POS of a word generally involves identifying those for other words in the same phrase, the correct meaning of a word is usually suggested by the ones surrounding it in the text, as in the two example phrases above where account and river are hints about the respective senses of bank.

Word sense disambiguation (WSD) is the specific task of identifying the correct sense for each ambiguous word in a text [84]. This task generally requires the availability of an external knowledge base to be used as a *sense inventory*, telling which are the possible senses of each word, which may differ across different sources; anyway, research on unsupervised discrimination (clustering) of different word senses exists. This task may be regarded as similar to POS tagging, but different methods are often employed.

Intuitively, sense disambiguation is generally based on the context in which every word appears, which usually consists into the surrounding words rather than the sentence containing it or even the whole document. Specifically, it is common to resolve ambiguities for multiple words at once, as the most likely sense for a word often depends from senses of the others, so that the overall most likely combination must be found.

Various approaches exist to WSD, which can be subdivided according to the type of external knowledge sources required. Some methods are based on structured knowledge bases like dictionaries: a representative example is the Lesk algorithm, where senses for more words are chosen in order to maximize the number of common words occurring in the dictionary definitions of all of them [64]. Likely to other tasks, other methods are based instead on training a general model from a collection of text which is fully or partially already tagged with correct senses [63], but also methods learning from untagged text exist [126].

1.4.3 Other tasks

The following are some other specialized task which are sometimes needed to be performed.

- ***Parsing*** refers to inferring the structure each sentence, which can be usually represented as a *parse tree* where the root corresponds to the whole sentence, intermediate nodes to phrases and leafs to single words. For example, the sentence "the book is red" can be decomposed into the noun phrase "the book" and the verb phrase "is red", which can be in turn decomposed into the single words.

- ***Named entity recognition (NER)*** is used to find proper nouns in a text and identify to which specific person, place or other entity refer to [83]. This usually implies resolving ambiguous references, for example identifying a person of which only the last name is given.

- Recognition of ***pattern-identified entities*** refers to spotting some specific type of information from common patterns: for example a string containing a "@" character (e.g. "bob@example.com") could be tagged as being an e-mail address, while a number followed by one of some specific strings (e.g. "88 mph") may be identified as a physical quantity.

- ***Coreference resolution*** refers to locating words in a text referencing the same entity. For example, a pronoun is generally used to refer to a person or an object explicitly named in a previous phrase, constituting what is called an *anaphora*: coreference resolution methods must correctly link the pronoun to the noun.

Chapter 2

General Techniques and Tools for Text Mining

Research on automated processing of text documents has been active for decades and, as stated above, the theme is today of great interest. Throughout the literature, many recurring techniques proved to be effective in many situations: this chapter gives an overview of these techniques, with some focus on those employed for text classification.

2.1 Brief history

Research on automatic analysis of text dates back to several decades ago: its need was generally driven by the general necessity to efficiently retrieve specific knowledge from consistent amounts of information. Storage of information, primarily in written forms, is performed since hundreds and even thousands of years. As the amount of information grows, retrieving specific needed pieces of it naturally becomes more difficult, so any method to improve the efficiency of search across information sources has primary importance. Since shortly after their advent, the great potential support of computers in storage and retrieval of very large amount of information had started to be explored.

Information retrieval (IR) is the research field dedicated to this issue, which can be dated back to the 1950s: in [78] was first proposed the general idea of a statistical approach based on keywords to automatize the search

of pertinent information. The typical problem studied in IR is the selection of documents from a possibly large collection which are pertinent to some query given by a user. Important advances in this field were made in the 1960s, including the development of the SMART Retrieval System [105], from which the vector space model (see next section) and other important concepts emerged.

One of the need within IR systems was the indexing of documents, to improve the query response time. In [80] a concrete probabilistic technique for indexing of documents in categories is presented along with experimental evaluation of its accuracy: this and successive similar works [9, 35] can be seen as early examples of text categorization methods.

Until 1990s, experimental evaluation of information retrieval methods was usually performed on relatively small datasets, due to the lack of availability of large collections. To support this research field, in 1992 has been launched the Text Retrieval Conference (TREC) [47], whose goal is to encourage large-scale information retrieval by providing the necessary infrastructure for its evaluation.

2.2 Bag-of-Words representation

The main hindrance in automated processing of text is its *unstructured* form: normally, a computer sees a text as a sequence of characters, usually without any additional information about how words, sentences and sections are divided and what they represent. The grammatical structures which are usually easily recognizable by human readers can't be trivially understood by computers, especially due to semantic and even syntactic ambiguities.

While NLP techniques can aid in discovering the grammatical structure and even the meaning of analyzed text and building a structured representation of it, applying them to large volumes of data may be not feasible, especially in contexts like online classification of documents where responses must be given quickly.

One solution to this problem is to build a somehow approximated representation of the text under analysis, which can be extracted with limited complexity but retains enough information to handle the task to be accomplished.

The basic elements constituting a text document are *words*: each word in a text has a meaning and constitutes a building block for phrases and

complete sentences. The single words of a text can be extracted in a relatively trivial way, by splitting the sequence of characters constituting the text on spaces and punctuation signs: this process is known as *tokenization*.

The extraction of single words is usually the first step for NLP techniques, which are then able to detect sentence boundaries and find the function and the meaning of each word. Anyway, even without using such techniques, it is trivial to identify the set of distinct words found in a document and obtaining a count of occurrences for each of them: this information is used to represent the document as a *bag of words* (BoW).

A bag of words is in practice the *multiset* (a set which can contain multiple occurrences of elements) of the words contained in a text document, without considering their order and excluding other elements like punctuation marks. While this representation implies a significant information loss, it proves to be effective to accomplish many practical tasks with good accuracy. An example is text classification, where the most recurring words in a document can indicate quite straightforwardly which is the topic treated in the document, given that representative words of categories of interest are known.

To allow usual standard machine learning algorithms to handle a collection of documents, they must be represented as vectors using a global set of features, according to the *vector space model* (VSM). A set of bags of words can be translated straightforwardly into a set of such vectors, considering the union of all distinct words as the set of features and representing each document as a vector with the count of occurrences or some other kind of weight (as explained in the following) of each word. In this view, each document is actually seen as a vector in a multidimensional space, where each dimension corresponds to one of the words appearing throughout the whole collection of documents.

The whole set of document vectors can be seen as a *document-term matrix*, where rows and columns correspond to documents and words (terms) respectively. A matrix built in such way is generally largely sparse (many entries are 0), as usually the global number of distinct words in a collection of documents is very high, but each document only contains a small part of them.

While here is described how to build a basic, "raw" representation of a collection of documents as bags of words and vectors, countless variants exist on this scheme to improve the accuracy of the representation and address the high dimensionality problem. As exposed in the following,

other features rather than words can be considered and a subset of them can be selected to represent documents.

2.2.1 Cosine similarity

In many circumstances, some of which will be shown in the following, it is required to somehow compare one bag of words, in form of a vector, with another one in order to evaluate their similarity. In a vector space $\mathbb{R}^n$, it is generally possible to define a *similarity function* $\mathbb{R}^n \times \mathbb{R}^n \to [0, 1]$ which maps to two vectors a numeric measure of how much they are similar. Similarity functions are somehow the inverse of the more known *distance functions*, which map to two vectors a scalar value in $[0, +\infty)$: conversion from distance to similarity of two vectors can be done by any function mapping 0 to 1 (equal vectors) and very large values to very small ones (distant vectors), such as $similarity = \exp(-distance)$.

The distance measure generally mostly employed in vector spaces is the *euclidean distance*. However, considering vectors representing text documents where values represent the frequency of each term, the similarity between two documents should depend on the proportions between the different values rather than on their absolute values. For example, two documents dealing with the same topic but with very different lengths (say 100 words versus 10,000) would generally be represented with two vectors similar in their orientation in the space, but different in their length (or *magnitude*), thus their euclidean distance would be misleadingly significant.

For this reason, a suitable solution is to evaluate the *angle* between two vectors without considering their length: the smaller is the angle, the more similar are the documents. To obtain a similarity measure constrained in the range between 0 and 1, the *cosine* of such angle is considered, which is equal to 1 for a null angle and is non-negative if all the vectors are so. This measure, called *cosine similarity*, can be computed from the components of two generic vectors $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ as follows.

$$\cos(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} a_i} \cdot \sqrt{\sum_{i=1}^{n} b_i}}$$

As shown in the formula, the cosine similarity is in practice the dot product of the two vectors normalized by their lengths.

As an example, the point $\mathbf{a}$ in Figure 2.1 is closer to $\mathbf{b}$ than to $\mathbf{c}$ by

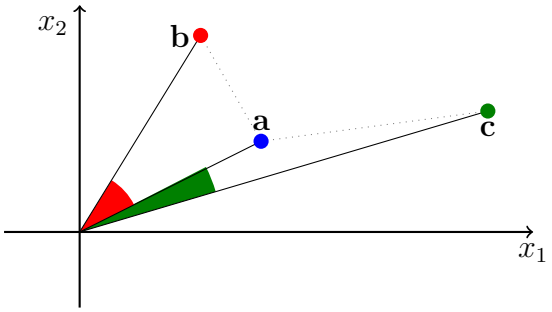

Figure 2.1 – *Comparison between euclidean distance and cosine similarity*

euclidean distance ($\|\mathbf{b} - \mathbf{a}\| < \|\mathbf{c} - \mathbf{a}\|$), but has an higher cosine similarity with $\mathbf{c}$ than with $\mathbf{b}$, because the angle between the two is smaller ($\cos(\mathbf{a}, \mathbf{c}) > \cos(\mathbf{a}, \mathbf{b})$).

Cosine similarity effectively expresses how much two bags of words are similar in the distribution of the words they contain, thus constituting a good estimation of much the topics they discuss overlap.

2.3 Extraction of features

In the need of reducing text documents to vectors, is important to define a set of predictive features which are effective in representing the original contents of the document. The trivial choice presented above is to consider words as they appear in the document, without considering their form or meaning. Variants of this approach aim to overcome this limitation by exploiting semantics of words, possibly using external knowledge. Other approaches consider other features than single words, such as n-grams.

2.3.1 Lemmas

Given a dictionary of a certain language (e.g. English), at each entry corresponds one *headword* or *lemma*, which is the canonical form representing a possibly wider set of words corresponding to the same entry. For many of such lemmas, *inflected* forms also exist, which express the same concept with differences in number, gender, tense or other categories. For example,

waiter and waiters refer to the same dictionary entry, having waiter as its lemma; similarly, served and serving are both inflected forms of (to) serve.

In the basic case presented above, where words are considered as meaningless tokens, different inflected forms of a same lemma are considered as distinct words. A different approach would be to represent each word by its lemma, without distinguishing its different forms. While this theoretically implies a loss of information, the advantage is a noteworthy reduction of the number of features which, in many cases, does not lead to a practical loss of accuracy. Taking again text classification as an example, it generally matters to know which lemmas appear and their frequency, while the specific form in which they appear (e.g. how many singular and plural forms of each noun are there) has negligible importance.

The application of this approach, however, is not straightforward, as a computer must be able to infer, for each distinct word encountered in a set of documents, its corresponding lemma. This process, known as *lemmatisation*, requires prior knowledge of the specific language under analysis. Such knowledge can be very complex, including a complete set of general rules addressing common cases (e.g. -s termination for plurals in many nouns and for singular third person in verbs) and enumerations of specific cases and exceptions to such rules (e.g. mice being plural of mouse).

2.3.2 Stems

In the most general sense, a *stem* of a word is a part of it. Here, *stem* is used to indicate the part of any word which is common to all its inflected variants. Some words have their stem equal to their lemma, but does not apply to any word. Recalling the examples above, waiter and waiters have waiter as their stem, which is also their lemma; however, served and serving have serv as stem, which is a truncation of the lemma (to) serve. In many cases, such as this latter example, the stem of a word, contrarily to the lemma, is not itself a word of the language (although a human reading a stem should generally be able to infer the lemma of the original word).

Likely to lemmatisation, *stemming* is the process of extracting the stem of an arbitrary word. Also in this case, the process is dependent from the language and requires proper knowledge of it. However, stemming a word is usually more simple than extracting its lemma: the stem is always contained in the word itself and can usually be extracted just relying on a not too many complex set of rules.

Given the efficiency of stemming algorithms, also known as *stemmers*, in the many situations where it is not necessary to have complete words as features, stems are instead used to efficiently identify groups of similar words.

Some stemming algorithms have been proposed, even since decades ago. The most commonly used among such algorithms is the *Porter stemmer* [97]. This algorithm, likely to others, is based on suffix stripping: a number of rules divided into some steps is defined so that each input word has its suffix removed or substituted according to rules whose preconditions match the word. For example, the -ing suffix is removed only if the resulting stem contains at least one vowel, so that serving is stemmed into serv, but sing is instead left as is.

2.3.3 n-grams and phrases

An *n-gram* is in general a sequence of n consecutive elements extracted from a wider sequence: common specific cases are for $n = 2$ (*bigrams*) and $n = 3$ (*trigrams*). Elements grouped in n-grams are generally either letters or words. For example, considering letters, from the word example can be extracted the bigrams ex, xa, am, mp, pl, le. Some works use n-grams as features in place of or in addition to single words, sometimes mixing different lengths.

n-grams of letters are not practical to use in place of words, as these serve usually to identify the topic discussed in a document and groups of letters (especially if short) would generally instead be poorly indicative of what the document discusses about. Instead, sequences of letters are usually employed in particular tasks where they can actually be effective as predictive features, such as classifying documents by language [16]: for example, English texts can be recognized by the high frequency of some bigrams such as th and er.

n-grams of words as features are instead more similar to words, as can be informative of the topic discussed in the text. The extraction of bigrams and trigrams can be useful to represent recurring compound expressions, such as text categorization or word sense disambiguation. A field where n-grams of words are currently particularly used is sentiment analysis, where some sequences of words expressing a specific polarity are present which would be not accurately represented as single words [85].

Some works also deal with the use of *phrases* as features, which may refer either to generic sequences of words which often recur in text – which reflects the above definition of word n-grams – or to *syntactic* phrases, which are those defined by the language grammar, having a meaning on their own. In [65] the use of syntactic phrases for text categorization is evaluated; it is observed anyway that phrases "have inferior statistical qualities" with respect to words, because distinct phrases are in higher number and many of them having similar meaning end up being split across many features.

2.3.4 Concepts

Types of features described above are extracted in a relatively trivial way from words, this with minimal processing of them. While words can generally give a good indication of the contents of the document, they anyway are not a perfect representation. A cause of this are the properties of synonymy and polysemy of natural language, implying that a meaning may be expressed by more than one word and a word, taken out of its context, may have more possible meanings. An ideal solution would be to represent a document with the *concepts* expressed in them, rather than (or in addition to) with the possibly ambiguous words.

Different methods exist to extract features of this type. Two general types of approaches can be distinguished: existing concepts can be extracted *statistically* from the documents or obtained from an external knowledge base. In the first case, the documents to be processed (or in some cases an external set of different documents) are analyzed in order to identify potential concepts according to co-occurrences between words in documents. For example, a set of words which (almost) always appear together in documents are likely to represent a unique meaning or very related ones: this set could be reduced to a single feature with no significant loss of information about the contents of documents. Some methods to extract *latent* semantic knowledge from collections of documents are discussed in Section 2.5.

The other possible approach is to use some sort of external knowledge base to obtain information about all the existing concepts and to correctly map words found within documents to them. Different sorts of knowledge bases can be used for this purpose, as discussed in Section 2.6, while in Section 5.2 are summarized some concrete methods for text categorization making use of external knowledge bases for extracting semantic features from documents.

2.4 Term selection and weighting

As seen above, there is more than one possibility to extract predictive features from a collection of text documents, with single words (or stems thereof) being a common choice given the good balance between accuracy and efficiency. However, an issue to be generally faced with any of these methods is the high number of global features, which potentially leads to poor performances in subsequent steps of the analysis. In general data mining, this is known as the *dimensionality problem*. What usually happens in practice is that each single document contains a manageable number of words or other features (say hundreds), but the union of all documents in collections of typical sizes brings a very high number of distinct terms. In this phase it is generally useful to apply techniques for *feature selection* to reduce the global number of distinct features to a smaller, convenient set. This step, if correctly tuned, can enhance the efficiency of the analysis with small or negligible loss of accuracy, which might be even improved in some cases.

Once a set of features has been defined, each document must be represented as a vector with one value for each of these features: a criterion to assign these values is necessary. *Term weighting* is the general task where, within a set $\mathcal{D}$ of documents, fixed a set $\mathcal{T}$ of features, a *weight* is decided for each term $w \in \mathcal{T}$ in each document $d \in \mathcal{D}$.

In the following, some recurring methods for selecting and weighting terms are presented.

2.4.1 Basic word filtering

Referring here to the case where single words are used as features, there are few trivial expedients which can be exploited while parsing documents to reduce the number of features.

An important aspect of the bag-of-words approach is that, while information about the presence and usually the frequency of distinct words is maintained, their position and their function in the text is lost. As words are stripped from their context, some more information becomes useless and may be discarded as well.

For example, while the first word of each sentence in a text is generally written with an uppercase initial, when the text is reduced to a bag of words it is no more important to distinguish sentence-opening words with an

uppercase letter. This is the main motivation for *case-folding*, i.e. turning all letters of all words to the same case (conventionally lowercase). This reduces the number of distinct words because, for any word that appears both in lowercase and with uppercase initial (and possibly all uppercase or in other forms), only one feature is extracted rather than two (or more).

As each word is taken without considering its context, it should have a meaning by itself to be somehow informative on the document it is found in. The words which better represent the content of a document by themselves are generally some nouns, verbs and adjectives which express the most relevant concepts of the text. On the other side, words like articles, pronouns and prepositions are quite common and do not express concepts by themselves, so are less likely to be useful as features. An example is the determinative article the: this is likely to appear multiple times in every document of a collection in English language, but it does not mean anything by itself and does not help in representing the semantic content of the document. For this reason words like these, called *stopwords*, are usually filtered out while documents are parsed. The removal of stopwords requires to be able to recognize them: generally a language-specific list is given. There isn't a unique list of stopwords for English or any other language, but all available lists are similar and using one rather than another usually has negligible effects on the results.

2.4.2 Feature selection

As the dimensionality problem affects data analysis in various fields, among which text mining is a prominent example, general techniques to automatically select an optimal subset of features have been devised. General *feature selection* algorithms are usually based on statistical observation of the data, regardless of what features actually represent.

Generally, analyzing a set of vectors, the features which can be discarded without affecting accuracy are those whose variability is too low (a feature with a constant value being the limit case) or too high. Also, if the values of two or more features are highly correlated to each other, only one of them is useful. These are *unsupervised* approaches, applicable in any learning task.

A trivial selection technique of this type is based on *document frequency*: features appearing in less than a set number N of documents are removed. This serves to ignore some very rare features which can be usually ignored without significant loss of information, like rather uncommon

terms or words with typos. This threshold N is usually very small, roughly near to the 0.1% of the number of documents, but even with such values a consistent number of unimportant features can be usually removed.

If additional information about the documents under analysis is available, it can be exploited to improve the efficacy of the feature selection process. Specifically, if documents are organized in categories, features can be evaluated accordingly to their correlation with them. This possibility will be discussed in §3.4.2, when dealing with text classification.

2.4.3 Term weighting

Once a set of features to represent documents as vectors has been decided, the vectors themselves must be computed: different *term weighting* schemes exist to determine the values to be assigned.

The weight of each term in each document should denote its *importance* in representing the contents of the document itself. The most basic weighting schemes just follow this intuition, so that weights of any document are assigned only on the basis of that document itself: these are referred to as *local* schemes. Below are briefly discussed the most common schemes.

The most basic scheme is *binary weighting*, which assigns 1 to any term present in the document and 0 to the rest of terms in $\mathcal{T}$. This method is necessary with learning algorithms which only accept presence of features and not weights, like some forms of probabilistic classifiers (see §3.5.1). Despite the fact it discards the information about the number of occurrences of each term, this method often yields practical results very similar to those obtained with other weights.

If the number of occurrences $\#(t, d)$ of each term t for any document d is instead to be considered, the simplest solution is to use this number itself as the weight of each term: this value (and the weighting scheme using it) is referred to as *term frequency* (*tf*). Intuitively, multiple occurrences of a term in a document are equally or more important than single appearances (this is also known as the *tf assumption*), so giving higher weights to terms appearing more often in the document may help to better characterize them.

A commonly employed variation of this scheme is *logarithmic term frequency*, where the number of occurrences is dampened using the logarithm function, with 1 added to the argument to maintain weight 0 on absent terms. This method can be employed to avoid overweighting the most re-

curring terms at the expense of possible other ones which are less frequent but still important.

In order to avoid giving more bias to long documents, the term frequency may be normalized by dividing all counts of occurrences by their sum. Although this simple normalization scheme is not much frequently used, it will be applied in experiments described later in the thesis.

Weighting schemes presented above compute the importance of each term locally to each document, without considering the larger context which is the collection of documents. *Global* weighting schemes measure instead the relevance of a term across the whole collection.

The by far most common global weighting scheme is *inverse document frequency* (*idf*), for which various similar formulations exist. The idf of a term t is usually computed as the logarithm of the inverse ratio of the number of documents n_t where t appears with respect to the total number N of documents in the collection. Another variant, commonly labeled as *probabilistic*, the total number of documents is substituted with the number of those where the term does not appear.

This type of schemes follows the hypothesis (also known as *idf assumption*) that terms appearing in many documents are *not* more important than more rare terms. Intuitively, the presence of a term appearing in many of the documents is not very informative: it could be for example strongly related to the topic treated in the whole collection and thus fail to be a distinctive feature for a limited amount of documents.

In summary, local and global weighting schemes, of which a short list is given in Table 2.1, assign convenient weights to terms in documents according to their importance measured respectively in the single documents and in the whole collection. These two aspects are complementary, but both are important to consider when creating the vectors for each document.For this, in most works, the weighting scheme adopted is the product of a local and a global one, so that each term is weighted with two factors denoting its importance in the two different aspects. The most common choice is to combine the basic term frequency with the inverse document frequency, to obtain the well known *tf.idf* weighting scheme.

$$tf.idf(t,d) = tf(t,d) \cdot idf(t) = \#(t,d) \cdot \log \frac{|\mathcal{D}|}{|\{\delta \in \mathcal{D} : \#(t,\delta) > 0\}|}$$

Table 2.1 – *Summary of local and global term weighting schemes*

Scheme	Formula
Local (weight within document)	
binary	$bin(t,d) = \min(1, \#(t,d))$
term frequency	$tf(t,d) = \#(t,d)$
logarithmic term frequency	$logtf(t,d) = \log(1 + \#(t,d))$
normalized term frequency	$ntf(t,d) = \frac{\#(t,d)}{\sum_{\tau \in \mathcal{T}} \#(\tau,d)}$
Global (weight within collection)	
inverse document frequency	$idf(t) = \log \frac{N}{n_t}$
probabilistic inverse document freq.	$idf_{prob}(t) = \log \frac{N - n_t}{n_t}$

$\#(t,d)$ = number of occurrences of term t within document d • n_t = number of documents where t appears • N = total number of documents

To maintain final values in vectors in a $[0,1]$ range and in order to avoid giving more bias to longer documents, sometimes the vector of each document is normalized according to some scheme, as suggested above for the term frequency. The most common one is *cosine normalization*, which in practice transforms the vector so that its length (or magnitude) is brought to 1 without changing its direction, which is often the actual important property of a bag of words, as discussed in §2.2.1.

$$w_{\text{normalized}}(t,d) = \frac{w(t,d)}{\sqrt{\sum_{\tau \in \mathcal{T}} w(\tau,d)^2}}$$

Also some supervised term weighting schemes have been proposed, which are based on known organization of documents in categories and are therefore mostly used in text categorization: a couple of related works are cited in §3.4.2 together with supervised feature selection.

2.5 Extraction of latent semantic information

Ideally, features extracted from documents should uniquely represent high-level concepts and topics in order to give an accurate representation of their contents. In practice, words or other features are generally far from the ideal condition due to the fact that there are more words are frequently used in each topic and each word may be used in more than one of them. From the point of view of the language, words can have multiple meanings (*polysemy*) and the same concept can be recalled by more words (*synonymy*). Additionally, some words may have different meanings but be somehow related.

These properties sometimes dmake ifficult to correctly spot similarities and correlations within data. For example, two documents may discuss the same topic using different words, so that they appear to be unrelated to each other. However, the relevant words of both documents may significantly occur simultaneously in either few or many other related documents: this information might serve to infer that all those words are somehow semantically related, thus the two example documents are potentially related despite they differ significantly in the words they contain.

Solutions exist based on the use of external knowledge bases, which are described in the following section. However, another possible approach is to analyze the available documents to recognize recurring dependencies between words, which are usually indicative of relatedness between them. These techniques to extract *latent* semantic information from documents are based on statistics and probability and are used across different text mining and general information retrieval applications.

2.5.1 Latent semantic analysis

Latent semantic analysis (LSA) [30], also known as *latent semantic indexing* (LSI), is a general technique to analyze relationships between documents and terms in a collection, extract high-level concepts and transform the representation of documents according to the identified relationships.

Summarily, LSA transposes documents of a collection and terms therein in a latent feature space, where dimensions ideally correspond to high-level concepts or components. Therefore, each document is represented as a

weighted mix of such components, while each term may similarly be related with different degrees to more concepts. This scheme is very similar to *principal component analysis*, which is used to map a vector space with possible correlations between dimensions to another space without such correlations.

Given a collection with n documents and m distinct terms extracted from them, in order to apply LSA, a $m \times n$ *term-document matrix* $\mathbf{X}$ must be built, with each cell $x_{i,j}$ containing the weight of term t_i in document d_j. Columns of $\mathbf{X}$ correspond in practice to bags of words for documents, with terms weighted according to some scheme: those presented above in 2.4.3 can be used, although different schemes based on entropy are often effective in this case.

Within this matrix, dot product (or cosine similarity) can be computed between two rows (terms) or two columns (documents) to estimate their correlation. A whole correlation matrix for terms or document may be obtained computing $\mathbf{X}\mathbf{X}^T$ or $\mathbf{X}^T\mathbf{X}$ respectively.

On the term-document matrix is applied *singular value decomposition* (SVD), a mathematical technique which computes a decomposition of the original matrix $\mathbf{X}$ into three matrices.

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^T$$

Of the resulting matrices, $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices sized $m \times r$ and $n \times r$ respectively, while Σ is a $r \times r$ diagonal matrix containing eigenvalues. The rationale is that each of the r eigenvalues corresponds to one of the aforementioned high-level components traced in the collection of documents and denotes how much it is relevant throughout the collection.

Eigenvalues are sorted along the diagonal of Σ in decreasing order, so that the ones coming first are related to the most relevant components. This allows to easily cut off less important components to a number $k \leq r$, simply by removing relevant rows and columns in the matrices. This reduction potentially allows to remove noise in the data, which can be constituted for example from terms or groups thereof appearing in few documents and poorly related to other ones.

Once such a value k is set, it can be considered to build an approximated version of the original term-document matrix $\mathbf{X}$, by multiplying the three reduced matrices: the resulting matrix $\mathbf{X}'$ will have its rank reduced from r to k. $\mathbf{X}'$ is structurally identical to $\mathbf{X}$ (its rows and columns are repre-

sentative of the same terms and documents as $\mathbf{X}$), but term weights are corrected so that noise is removed and evident correlations between terms (or between documents) are accounted. For example, if two terms t_a and t_b frequently occur together in documents, a document containing only t_a of the two will anyway have a weight for t_b higher than zero (and vice versa).

From the reconstructed matrix $\mathbf{X}'$ or directly from the truncated matrices used to compute it, similarity between terms and between documents can be computed according to the corrected weights, which will generally be different from the corresponding one computed from the original matrix. In the common case where documents most related to a query must be found, using the common approach where the query is represented like a document to be compared to known ones, it should first be mapped into the latent space to undergo the same correction of values: this procedure is known as *fold-in*. In the latent space, related documents which do not contain the exact words of the query but strictly related ones can be found.

2.5.2 Probabilistic models

The LSA technique described above is based on singular value decomposition, which assumes a normal distribution of weights in the term-document matrix: this modeling is not fully accurate, although particular weighting schemes can make it work better. For this, improved techniques have been proposed, based on different probability models, in particular on multinomial models, which better represent the occurrences of words in documents.

A first extension of the basic LSA technique has been the *probabilistic latent semantic analysis* (PLSA) [49], which considers a probabilistic model based on an hidden class variable $z \in \mathcal{Z}$, which correspond to components (dimensions of the latent space) in LSA. In practice, each word and each document under analysis are considered to have affinities to these latent classes, which ideally represent topics, each with specific recurring words. From this, the occurrence of a word w in a document d is seen as a mixture of these classes; in another parameterization, a document is seen as a mixture of classes, which are in turn seen as mixtures of words.

$$P(d, w) = \sum_{z \in \mathcal{Z}} P(z)P(d|z)P(w|z) = P(d) \sum_{z \in \mathcal{Z}} P(z|d)P(w|z)$$

The conditional parameters of the model are estimated by the Expecta-

tion Maximization algorithm. While PLSA constitutes a better probabilistic model than the one assumed in LSA for a corpus of existing documents, it has the shortcoming of not providing a *generative* model able to represent any document, even outside of the corpus.

A further extension of PLSA, *latent Dirichlet allocation* (LDA) [5] addresses this issue by providing a probabilistic model for generation of documents: it assumes the mixture of topics $\mathbf{z}$ conditioning words $\mathbf{w}$ of a document to have a Dirichlet-distributed prior θ, which is parameterized by a vector α. Another parameter of the model other than α is the per-topic word distribution β. This gives the following joint distribution for a document, where N is the number of words (assumed as constant here for simplicity).

$$P(\theta, \mathbf{z}, \mathbf{w}|\alpha, \beta) = P(\theta|\alpha) \prod_{n=1}^{N} P(z_n|\theta) P(w_n|z_n, \beta)$$

Further improvements and extensions of these models exist, such as the *pachinko allocation model* [70], which captures arbitrary correlations between topics which can be represented as a directed acyclic graph.

2.6 Linguistic and semantic knowledge bases

Techniques described in the previous section are used to infer semantic information about words present throughout a collection of documents under analysis basing on statistical co-occurrence within the collection itself. Obtaining information about semantic relatedness between words from the collection itself can be advantageous, as the words for which information is obtained are exactly those for which is needed, independently from the domain. However, extracted information may happen to be incomplete or even erroneous, due to statistical anomalies in the collection.

Potentially, more accurate knowledge can be obtained from an external base of knowledge about the language under analysis. In general, a complete enough knowledge base should be used, in order to grant coverage of the information needed in the domain of the collection under analysis.

Different resources have been used as knowledge bases for text mining applications, including some which were not initially built with this specific

purpose. At a very high level, two types of external knowledge bases can be generally distinguished.

- *Structured* knowledge bases are specifically built to be machine-readable resources employable in specific text mining and natural language processing task. These can be subdivided according to the type of information they provide, roughly ranging from *dictionaries* or *lexicons* to *ontologies*. While the former are basically lists of words possibly with descriptions associated, the latter are usually large sets of concepts with various types of links across them.

- *Unstructured* knowledge bases are generic collections of data, usually in form of text primarily built for consultation by humans, from which structured knowledge can be somehow extracted. Sometimes, these consist into collections of documents different from those under analysis, from which additional knowledge can be extracted. Some of these collections are also partly structured, including for example small structured parts and/or links between documents: a prominent example is Wikipedia, which has often been used as a source of semantic knowledge.

The proper use of such external knowledge bases allows to more or less accurately to "understand" words and assign them meanings, which can be possibly linked with each other. This allows for example to represent a document by the concepts it contains, rather than by possibly ambiguous words, as discussed in §2.3.4. An example of task for which the use of knowledge bases is often important is word sense disambiguation (§1.4.2), as they provide possible senses for each word and can be also used to support the task of disambiguating between them. Methods for text categorization also exist relying on these resources: some of them will be summarized in Section 5.2.

2.6.1 WordNet

Throughout the text mining literature, one of the most employed knowledge bases is WordNet, which has been subsequently extended in many forms.

WordNet is a lexical database of the English language, created in the Cognitive Science Laboratory of Princeton University [81]. WordNet stores

more than 100,000 English lemmas, divided into four parts of speech (POSs, see explanation in §1.4.1): nouns, verbs, adjectives and adverbs. Each lemma appears in one or more *synsets* (contraction of *synonym sets*), also divided into the same four POSs: each synset represents a concept described by a human-readable *gloss* and includes one or more lemmas which express that same meaning (which are thus *synonyms*), possibly with only slightly different acceptations. The possibility for one synset to include more lemmas reflects the *synonymy* property of the language, while the fact that one lemma may appear in more than one synset reflects *polysemy*.

Other than lemmas and synset, an important part of WordNet is constituted by the relationships existing between them. Two types of relationships are defined within WordNet: each *semantic* relationship involves two synsets, while each *lexical* relationship involves the specific instances of two lemmas appearing in two distinct synsets, which in WordNet are formally referred to as *words*. Relationships are represented internally as *pointers*, which start from a synset or word and point to another one which is somehow related. Each pointer has a type, which indicates in which mode the two objects are related; 29 different pointer types exist in WordNet.

The following are the most recurring types of pointers which can be associated to a synset X.

- A *hypernym* is a synset Y which represents a more general concept with respect to X, whose meaning is included in Y. This relationship can be intuitively expressed with "X is a (type of) Y", as in for example "car is a vehicle" or "dog is a type of animal". Hypernymy exists between nouns and between verbs and creates a *subsumption* hierarchy of concepts in form of a directed acyclic graph, where most synsets have exactly one hypernym (so almost tree-like). Nouns are organized into 25 hierarchies (*animal*, *food*, *location*, *feeling*, ...), merged at the top levels by generic concepts (*unique beginners*), having root in the synset entity.

- A *hyponym* is a synset Y representing a more specific concept than X. Hyponymy is exactly the inverse relationship of hypernymy: Y is hyponym of X if and only if X is hypernym of Y.

- An *instance hyponym* is a synset Y which represents a single specific object which is of the type represented by X. For example, Sun is an instance hyponym of star.

- An *instance hypernym* is the type of entity of which X is an instance. Instance hypernymy is the opposite of instance hyponymy.
- A *meronym* is a synset Y representing an object which is somehow part of X. Three types of meronymy are distinguished.
  - A *part meronym* is a constitutive part of the object X, necessary to obtain it (e.g. wheel is part meronym of car).
  - A *member meronym* is a member of the set X (e.g. player is member meronym of team).
  - A *substance meronym* is an object made of X as material (e.g. glass is substance meronym of bottle).
- A *holonym* is a synset Y representing an object of which X is a part. Holonymy is the opposite of meronymy and has the same three distinctions (*part holonymy*, *member holonymy*, *substance holonymy*).

The following are instead the most recurring types of pointers which a word x may have.

- An *antonym* is a word expressing an opposite meaning with respect to x. Antonymy is a reflexive relationship. For example, male and female are antonyms of each other.
- A *derivationally related form* is a word with the same root or stem. This also is a reflexive relationship. For example, fish (the animal) and fishing are derivationally related.

From WordNet, many extensions and related projects have been developed. Some of such extensions consist in augmentations of the original data with additional information. An example of this is *SentiWordNet*, where to each synset weights are assigned indicating whether it expresses positivity (e.g. wonderful), negativity (e.g. awful) or objectivity (e.g. complex) [3]: this is useful in opinion mining tasks.

Some projects are focused instead on creating equivalent databases of different languages, possibly with links across them. *EuroWordNet* is a system of wordnets in multiple languages, interconnected by an *Inter-Lingual Index* [117]. Between examples of projects focused on single languages, *MultiWordNet* is a database for the Italian language aligned with the original English one [95].

Other projects aim to interlink WordNet data with other resources and to integrate it into the Linked Open Data network constituting the Semantic Web. A notable project is *BabelNet*, a multilingual ontology created by automatically linking WordNet to Wikipedia.

The original WordNet, being already widely used in many tasks, has been chosen as the reference knowledge base for the text categorization method presented in Chapter 5 based on semantic relationships between words.

Chapter 3

Text Categorization

Here the general task of *text categorization* is described, distinguishing its variants and reporting some known solutions from relevant literature.

3.1 Problem description

Generally speaking, *text categorization* (or *classification*) is the task of labeling each text *document* of a set with *categories* (or *classes*, as commonly named in general data classification). Categories assigned to each document must be a subset of a priorly defined set of known categories, in contrast e.g. to *text clustering*, where no such set is given.

Throughout the thesis, the global set of documents of a particular collection is usually denoted with $\mathcal{D}$, while $\mathcal{C}$ denotes the set of known categories with which documents can be labeled. Given this notation, the general goal of text classification can be formalized as giving in output a *labeling* $\hat{\mathcal{L}} : \mathcal{D} \times \mathcal{C} \to \{0, 1\}$, indicating for each couple $(d, c) \in \mathcal{D} \times \mathcal{C}$ whether d should be labeled with c (1) or not (0).

The set $\mathcal{C}$ of categories is decided according to some practical criterion. In the most common case, each category represents a *topic* which is discussed in some documents in the collection: the need is to extract the subset of topics significantly treated by each of the documents.

Keeping valid the general scheme of the problem, categories may refer to something else other than topics. Works exist dealing with classification of documents by author or by language, for example. Anyway, these tasks

often require ad hoc techniques to achieve good accuracy and efficiency: for example, many works to classify text by language usually work with n-grams of letters instead of words, to have a smaller and possibly more significant set of features (§2.3.3).

There also are tasks which are not always regarded as "text classification", but are ascribable as such, usually for being more specific. For example, *spam filtering* in a mail inbox may be regarded as labeling each text document (a mail message) as belonging to either SPAM or NOT SPAM (or HAM, in jargon) category. In these cases, the techniques usually employed in text classification by topic can be (and often are) applied.

3.2 Variants

The description above is very general and refers to a wide range of works about text classification. Works usually give slightly narrower definitions of the problem they address, which are anyway similar and some of them can be seen as generalizations or specializations of others. Here are described the commonly possible variants of the general problem.

3.2.1 Binary, single-label and multi-label classification

One thing to define, other than how many and which categories can be globally applied to documents, is how many of them can be applied to each single document.

Generally speaking about classification of objects (either documents or not), the most basic task is *binary* classification, where each object must be assigned to one and only one of two possible classes. It is common here to denote one class with some identifier, say c, and the other one as its complement $\bar{c}$: in this point of view there is a single class to which each object may belong or not. This is the case of the spam filtering example cited above, as well as other similar filtering tasks. This type of problems fits well to some machine learning models which are natively oriented to binary classification, such as support vector machines (§3.5.2).

While binary classification considers only two possible classes, *multi-class* classification entails an arbitrarily sized set of classes: in this case, two possibilities are common. In *single-label* classification, each document

is assigned to one and only one possible category: in this case, considering classification by topics, we impose that each document must treat a single one of such topics, which may be reasonable or not according to the specific context.

With *multi-label* classification, instead, no constraints are set on the labeling of each document: given a set $\mathcal{C}$ of possible categories, to each document can be assigned an arbitrary subset of them, which is usually allowed to be empty. While some machine learning algorithms have been specifically proposed for multi-label classification problems (often being general-purpose adaptations of standard ones), most of them can only treat binary and single-label cases. The most common solution is to consider a multi-label problem with $|\mathcal{C}|$ possible categories as $|\mathcal{C}|$ distinct binary classification problems, each to determine which documents should be labeled with one of the categories. In this approach, the potential dependencies between categories are not considered, meaning that the probability for a document to belong to a certain category does not depend from its associations to the other categories. Although considering independence between categories may be theoretically not correct, it is usually an acceptable approximation in practice to ease the classification process.

3.2.2 Hierarchical classification

In the most general case, categories of the set $\mathcal{C}$ are independent from each other: the assignment (or absence thereof) of a certain category to a document does not depend from which of the other categories are assigned to that document. This happens when the scopes of different categories do not significantly overlap. Consider for example the classification of documents under three categories corresponding to topics COMPUTER, MUSIC and SPORTS: while some documents may be related to more than one of these categories at the same time (e.g. a document about a soccer video game might regard both COMPUTER and SPORTS), it is generally not possible to make assumptions about the likelihood for a document to belong to a category knowing whether it belongs to any other one.

Anyway, there is the possibility for topic categories to be inter-related by means of *is-a* relationships, meaning that some known topics are more specific branches of other topics, also represented by categories. For example, considering a set of known categories including SPORTS, BASEBALL and HOCKEY, documents talking about BASEBALL are forcedly also dealing

with SPORTS and the same is valid for HOCKEY. Given a set of topics presenting these relationships, a hierarchical taxonomy of these topics can be created, where some of them are recursively broken down into more specific branches.

Hierarchical text classification generally refers to classifying text document under hierarchically-organized taxonomies of categories. Each of such taxonomies is generally structured as a single-rooted tree, where each node, apart from a *root* node, represents a category having a distinct, single node as its parent. Each category may have any number of children categories, representing more specific topics: categories with no child nodes are *leafs* of the tree. As a generalization, a taxonomy may also be represented by a directed acyclic graph (DAG), which in practice allows multiple roots and nodes with multiple parents, but in the following, unless otherwise stated, only single-root trees are considered.

A hierarchical taxonomy can be expressed formally as a partially ordered set $\langle \mathcal{C}, \prec \rangle$, where $\mathcal{C}$ is the set of categories and $\prec \subset \mathcal{C} \times \mathcal{C}$ is the asymmetric, transitive *is-a* relationship. In practice, $c_d \prec c_a$ means that c_d represents a specific topic within the wider discussion area represented by c_a: in this case, c_a is said to be an *ancestor* of c_d, which is in turn a *descendant* of c_a. In this formalization, the (direct) *parent* of a non-root category c_d is the only category c_p satisfying $c_d \prec c_p \wedge \nexists \gamma \in \mathcal{C} : c_d \prec \gamma \prec c_p$, while *children* of a category c_p are those categories whose c_p is parent.

The use of a hierarchical taxonomy of categories is often useful to better organize documents, allowing to find specific ones starting from general discussion areas and progressively narrowing down the domain to the topic of interest. A typical example of this organization are web directories, where great numbers of websites are organized in a fine-grained taxonomy of categories which can be browsed by the user, from the home page presenting the top-level categories to the sub-pages of specific topics listing general related websites and possibly even more specific sub-categories where other websites are distributed.

The most extended and known web directory is the *Open Directory Project*, also known as DMOZ[1]: it is maintained collaboratively by users and contains millions of web links organized into a dynamic hierarchical taxonomy of about one million categories. Figure 3.1 shows a small part of the top of this taxonomy, which in some branches goes down for ten levels

[1] `http://dmoz.org`

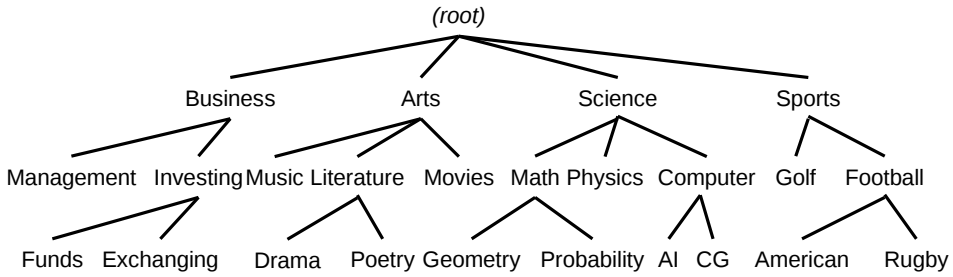

Figure 3.1 – *Excerpt of the DMOZ taxonomy of categories.*

and more.

Within this general representations, there are some different specific cases handled by hierarchical algorithms. Documents are often classified under a single category of the tree, as in single-label classification, but assuming implicitly also its ancestor categories up to the root; in some cases documents may be instead assigned to multiple, independent categories. Also, the problem (and the used method) might restrict classification of documents to leaf categories only or allow to label them also with intermediate categories. Some works consider that a document classified to a category c is also automatically classified to all ancestor categories up to the root: in this vision the problem must be multi-label, but in many cases classification is anyway restricted to a single path between the root and the most specific category.

With some exceptions, hierarchical classification is generally tackled taking into consideration the taxonomy of categories, usually by a top-down approach which progressively narrows down the possible categories of each document. Solutions specifically relevant to hierarchical classification will be discussed in Section 3.7.

3.3 Knowledge engineering approach

Before the establishment of the now common machine learning-based methods, many solutions for automatic classification of text documents were based on manual coding of the required knowledge by human experts. This

knowledge engineering approach was most common in operational settings.

The approach was to manually build a set of rules subsequently followed by a computer to classify documents. The accuracy of classification reached with this method was approximately at the same level of subsequent approaches based on machine learning, but reaching them required large amounts of human effort to write a sufficiently representative set of rules. This manual work had to be repeated every time the set of categories changes.

The mentioned rules have the general form $\{condition\} \Rightarrow \{category\}$, read as "if the document satisfies $\{condition\}$, then label it with $\{category\}$". Such conditions are usually given in disjunctive normal form, i.e. a disjunction (OR) of conjunctions (AND) of possibly negated clauses. Common primitive clauses refer to the presence of specific words in the document, likely to many later methods. Below is given a hypothetical sample of one of such rules.

$$(\mathsf{bat} \wedge \mathsf{ball} \wedge \neg\mathsf{softball}) \vee (\mathsf{pitcher}) \Rightarrow \textsc{baseball}$$

This rule is relatively trivial; in real scenarios with many categories, a consistent amount of more complex rules is generally needed to be compiled by knowledge engineers and domain experts. The machine learning-based methods emerged later are aimed in practice to obtain this knowledge in an automated way.

3.4 Machine learning approach

In the late 1990s, also due to increasing computational power of computers, machine learning-based methods for text categorization became widely common. In this approach, machine learning algorithms are used to infer one or more knowledge models from a *training set* of labeled documents; models can be used subsequently to predict correct categories for new documents.

An important limit of this approach is the need for a suitable training set: this must contain a consistent amount of documents as similar as possible to those to be classified and already labeled with the correct categories. In practice, if such set is not available, some documents must be classified manually to automatize the rest of the process. On the other side, once a training set is available, the required knowledge is extracted from it in a

fully automated way and can be as complex as needed to accurately classify further documents under the same categories.

In this approach to text categorization, documents are usually represented in a vector space using the bag-of-words representation described in Section 2.2, so that well-known standard machine learning algorithms can be used *off the shelf*, without the need to develop specific solutions.

3.4.1 General setup

Suppose that a set of documents $\mathcal{D}$ must be organized into a set $\mathcal{C}$ of categories. Machine learning-based text categorization starts from retrieving an adequate training set $\mathcal{D}_T$ of pre-labeled documents, which should be representative of documents to be classified: this entails using roughly the same words and being labeled coherently with the same categories. An ideal and possible case is that some of the documents of $\mathcal{D}$ are manually labeled to build the set $\mathcal{D}_T$. Anyway, a common assumption is that a suitable training set $\mathcal{D}_T$ is priorly available and that the set of documents to be classified is instead unknown, so that no knowledge can be extracted from it. Ideally, the distribution of categories of training documents should be as balanced as possible, or corresponding to distribution of subsequent documents to be classified.

Once a training set of labeled documents becomes defined, to apply standard machine learning algorithms, these must be represented as vectors in a common feature space. The typical choice is the general bag-of-words model, usually taking stemmed words as features; common preprocessing steps are case-folding and removal of stopwords. After applying feature selection and weighting, documents are reduced to the vectors constituting the training set.

The subsequent steps depend from the type of considered classification. In the simple case of single-label (possibly binary) classification, each training vector can be labeled with the single relevant category for the corresponding document. However, the more general case of multi-label classification has often been considered. As cited above, a practical solution is to consider one separate binary problem for each category, so that each document to be classified is included in or excluded from a category independently from the others. In this case, in practice, one binary classifier is trained for each category, always using the same algorithm and the same training set, but with “yes” and “no” labels differently assigned.

After one or more classifiers are extracted, new documents can be classified in the same categories. Each input document must be represented with a vector compatible to those of the training set: the same features selected for the training documents are extracted, with the possibility of losing some words (those which were not selected from the training set).

3.4.2 Supervised term selection

In Section 2.4 have been presented some common techniques to select a suitable set of features to represent documents. These techniques can be generally applied in any text mining task where a set of documents is given, regardless of any possible additional information about them. However, in text classification, the training set under analysis has a labeling associating documents to categories: this knowledge can be exploited during feature selection to improve it.

An ideal method is to select the features which are most helpful in determining the correct class of each object, even giving less consideration to unsupervised criteria. For example, given a set of documents with categories representing sports like BASEBALL and HOCKEY, the word injury would probably appear in an average number of them and would be a hint about the content of the text, but if it appears with similar likelihood in any category, it does not help in classifying the documents. On the other end, a word like batting gives a greater clue about the category of a document, even if it happens to be rare throughout the collection.

Different methods exist to compute numerically the relevance of a feature to determine the class of data, theoretically based on statistics and information theory: they are often based on the probabilities of (co-)occurrence of features and classes in the data. These probabilities are estimated from the training set, which is assumed to be representative of the domain under analysis.

An evaluation scheme used to perform feature selection usually computes a value for each feature-class couple $(t, c) \in \mathcal{T} \times \mathcal{C}$, indicating how much t is useful in determining the membership of documents in c. To obtain a general ranking of most predictive features, a single score must be given to each of them: possible solutions are to consider the maximum or the mean of scores for each class. From the ranking, a number of topmost entries can be selected as the set of features used to represent documents.

In this case, term selection is *global* because a single set of features is selected for the whole process. Anyway, as each category usually has its own representative words, it would be beneficial to consider them one by one. This is possible in the multi-label setting where one classifier for each category is trained: in this case, the models can be classified from training sets with different features, which are the best in each single case. This is referred to as *local* feature selection and usually brings to slightly better results.

Some schemes exist to evaluate the predictive power of a feature t within a class c: these are commonly defined on the basis of probabilities, sampled from the training set, for a document to contain (t) or not ($\bar{t}$) the term and to be labeled (c) or not ($\bar{c}$) with the category. Alternative but similar formulations are based on four variables conventionally denoted with A, B, C and D, indicating respectively the number of training documents

- (A) containing t and labeled with c,
- (B) containing t and not labeled with c,
- (C) not containing t and labeled with c,
- (D) not containing t and not labeled with c.

The total number of training documents is indicated with $N = A + B + C + D$.

Examples of used functions are the *pointwise mutual information* and the *chi-square* (χ^2) statistic, expressed hereafter in the two alternative modalities.

$$\mathrm{PMI}(t,c) = \log \frac{P(t,c)}{P(t)P(c)} = \log \frac{N \cdot A}{(A+B)(A+C)}$$

$$\begin{aligned} \chi^2(t,c) &= \frac{|\mathcal{D}_T|(P(t,c)P(\bar{t},\bar{c}) - P(t,\bar{c})P(\bar{t},c))^2}{P(t)P(\bar{t})P(c)P(\bar{c})} \\ &= \frac{N \cdot (AD - CB)^2}{(A+C)(B+D)(A+B)(C+D)} \end{aligned}$$

Some studies [124, 38] compare how different schemes perform in text categorization tasks: information gain and chi-square generally result among the best choices.

Other than in feature selection, knowledge of how documents are labeled with categories can be exploited in term weighting. In [29] is proposed to use the schemes already used to evaluate which term are selected also as global weights for them, possibly substituting the classic *idf* scheme. In [60] a *relevance frequency* factor is instead proposed to replace *idf*, based on the ratio between variables A and B discussed above.

3.5 Common learning algorithms for text

To train a classifier on vectors extracted from text documents, theoretically, any standard supervised machine learning algorithm may be used to infer knowledge models to classify text documents suitably represented as vectors. Anyway, throughout the literature, some specific approaches have proven to work sensibly better than others, often with good efficiency. Here are described in detail two classes of general supervised learning models often used in text categorization, then other relevant approaches are summarized.

Throughout the discussion, single data objects are referred to as *instances*, while *class* is used to denote one of some possible partitions of the data. These classes may not exactly correspond to the categories: for example, in a multi-label problem decomposed into multiple binary classification problem, each problem related to a category c has two complementary classes: documents to be labeled with c and documents not to be labeled with it.

3.5.1 Naïve Bayes

Naïve Bayes generally refers to a family of learning algorithms based on probability theory, especially on the well known *Bayes theorem*, relating belief conditioned by an evidence to previous belief.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

This is the most basic and known form of the Bayes theorem, stating that the *posterior* probability of an event A conditioned by an event B is the product between the *prior* probability of A and the support to A provided by B.

The rationale of a naïve Bayes classifier is to infer the support given by each feature to each class, assuming it to be completely independent from the other features: this is what makes the approach "naïve". Ideally, a classifier should compute the likelihood with which an instance should be labeled with class c from the combination of values of features $t_1, t_2, \ldots, t_n$.

$$P(c|t_1, t_2, \ldots, t_n) = \frac{P(c)P(t_1, t_2, \ldots, t_n|c)}{P(t_1, t_2, \ldots, t_n)}$$

However, considering the possible combinations of values of an even moderately large number of features, an exact computation is unfeasible, so the "naïve" assumption of complete independence between features is used. This assumption allows to rewrite the formula in a tractable form.

$$P(c|t_1, t_2, \ldots, t_n) = \frac{P(c)\prod_{i=1}^{n} P(t_i|c)}{P(\prod_{i=1}^{n} P(t_i))} \propto P(c)\prod_{i=1}^{n} P(t_i|c)$$

The denominator just serves as a normalizing constant, so the most likely class c for an instance can be assumed to be the one yielding the highest numerator. Through normalization, proper probability estimations can be made, which can be useful in evaluating the obtained classifier (see §3.8.2).

To avoid numeric underflow during computation of the product, its logarithm can be computed instead, allowing to decompose it in the sum of those of each factor, thus obtaining a linear formula.

$$\log P(c|t_1, t_2, \ldots, t_n) \propto \log\left(P(c)\prod_{i=1}^{n} P(t_i|c)\right) = \log P(c) + \sum_{i=1}^{n} \log P(t_i|c)$$

Summing up, the parameters of the classification model are, for each class c, the prior probability $P(c)$ and the conditional likelihood $P(t|c)$ for each feature t. The prior probabilities can be estimated from the training set itself as the ratios of instances belonging to each class, assumed to be equally distributed ($P(c) = \frac{1}{n} \forall c$ where n is the number of classes) or be set to other arbitrary values based on some additional knowledge of the context.

For what concerns the $P(t|c)$ parameters, always estimated from the training set, their computation depends from the possible values which can

be assumed by each t and the model assumed for them, for which some possibilities exist.

In the basic case with binary-valued features, which in text categorization refer to the presence or absence of terms in each document, each parameter $P(t|c)$ refers to the estimated probability of term t to either appear or not (according to the case under analysis) in class c. Denoting with $p_{t,c}$ the probability estimated from the training set of t appearing in c and with $x_t \in \{0, 1\}$ a sampled value of feature t, the related parameter can be written as the distribution between the two cases.

$$P(t|c) = p_{t,c}^{x_t} \cdot (1 - p_{t,c})^{1-x_t}$$

Should be noted that, if a term t never appears within a class c, the probability $p_{t,c}$ would be estimated to 0, implying $P(t|c) = 0$ and consequently $P(c|t, \ldots) = 0$ for any instance containing the term. To avoid this, *Laplace smoothing* is usually applied: the counts of occurrences of terms in classes are increased by one.

Considering instead each feature as the number of occurrences of some term (*term frequency* weighting), a *multinomial* model is often used where $P(t|c)$ must account for all possible values.

Despite the usually strong approximation brought by the independence assumption, naïve Bayes classifiers have been largely employed in information retrieval applications [66] and, combined with optimal pre-processing methods, have proven to be moderately effective in text categorization tasks, although bringing lower accuracies compared with other methods [123], also because of susceptibility to unbalanced distribution of training classes.

Given its efficiency and ease of implementation, the naïve Bayes classifier is a common choice as a baseline method to be compared with new proposed ones or as a learning algorithm to be used as an alternative to others in more complex classification methods. However, some works tested improvement on the basic method with the goal to obtain better results: this has been done for example by suitably normalizing the weights (likelihood parameters) of the model [57], by partially relaxing the standard complete independence assumption [93] or by combination of some of such improvements [102].

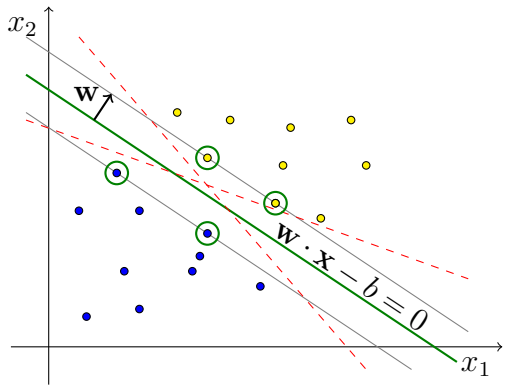

Figure 3.2 – *Example of SVM operation: yellow and blue points are training data points of class 1 and -1 respectively, the green line is the computed separation hyperplane, circled points are support vectors, red dashed lines are sub-optimal separation hyperplanes*

3.5.2 Support Vector Machines

Support vector machines (SVM) are a class of supervised learning methods to infer binary classification models. The general idea is to infer, from a training set of points of a high-dimensional space divided into two classes, a hyperplane or a set thereof which is as most effective as possible in correctly separating points of the two classes.

Formally, considering a m-dimensional vector space, a usual SVM algorithm accepts as input a set of points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n \in \mathcal{X} = \mathbb{R}^m$ along with respective class labels $y_1, y_2, \ldots, y_n \in \{-1, 1\}$. The output is a hyperplane, representable as a set of points $\mathbf{x}$ satisfying $\mathbf{w} \cdot \mathbf{x} - b = 0$, where $\mathbf{w}$ is the vector normal to the hyperplane itself.

If points of the two classes are linearly separable, the parameters $\mathbf{w}$ and b can be defined so that all positive (class 1) points lie in the subspace of points $\mathbf{x}$ having $\mathbf{w} \cdot \mathbf{x} - b \geq 1$, while all negative (class -1) points lie in $\mathbf{w} \cdot \mathbf{x} - b \leq -1$. If the parameters are optimized to maximize the distance between the two hyperplanes $\mathbf{w} \cdot \mathbf{x} - b = 1$ and $\mathbf{w} \cdot \mathbf{x} - b = -1$ (equivalent to minimizing $||\mathbf{w}||$), some training points will lie on the hyperplanes themselves: these are the *support vectors* defining the position of the optimal hyperplane separating the two classes.

The *soft margin* variant [21] takes into consideration the possibility that

input data is not linearly separable, due for example to labeling errors or outliers: in this case, the generated model will unavoidably mislabel some training points, but this can be acceptable as long as the computed hyperplane is as most effective as possible in correctly classifying new points. A *cost* parameter c is introduced, controlling the penalty to be given to misclassified training points. If its value is high, these misclassifications are minimized, but this could lead to choose a sub-optimal hyperplane which does not properly generalize the training data (this phenomenon is known as *overfitting*). On the other side, a low value of c allows more misclassified training instances to find an hyperplane with a higher distance from the remaining training points.

In its basic formulation, SVM can separate points in two classes only linearly. An intuitive solution to overcome this limitation is to set a non-linear mapping $M : \mathcal{X} \to \mathcal{V}$ from the original space $\mathcal{X}$ to a new one $\mathcal{V}$ and run the SVM training algorithm in the latter. However, transforming vectors coordinates across the two spaces can be very inefficient. To enable non-linear separation while maintaining the algorithm efficient, the *kernel trick* can be employed: instead of explicitly mapping points to $\mathcal{V}$ and compute their dot products (or more generally *inner products* $\langle \cdot, \cdot \rangle$) in it, an appropriate kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is used which takes vectors in the original space as input and returns their inner product in the transformed space [10].

$$\forall \mathbf{a}, \mathbf{b} \in \mathcal{X} : k(\mathbf{a}, \mathbf{b}) = \langle M(\mathbf{a}), M(\mathbf{b}) \rangle_{\mathcal{V}}$$

The best kernel function to use (if any) largely depends on the nature of the data to be classified. The following are some commonly employed kernel functions (in parenthesis, the parameters of each).

- Polynomial (degree d, c): $k(\mathbf{a}, \mathbf{b}) = (\mathbf{a} \cdot \mathbf{b} + c)^d$ (using $d = 1$ and $c = 0$ entails no mapping at all)
- Gaussian radial basis function (γ): $k(\mathbf{a}, \mathbf{b}) = \exp(\gamma ||\mathbf{a} - \mathbf{b}||^2)$
- Sigmoid kernel (κ, c): $k(\mathbf{a}, \mathbf{b}) = \tanh(\kappa \mathbf{a} \cdot \mathbf{b} + c)$

Additionally, kernel functions can be defined in domains different from real vector spaces, in order to apply SVM to different types of data without performing feature extraction. For example, *string kernels* measure similarity

between sequences of symbols, while *graph kernels* exist operating on nodes of a graph or on whole graphs.

The first notable use of support vector machines for text categorization is reported in [52], where arguments are given based on usual properties of text: high dimensionality, sparsity and (often) linearly separable categories. Successively, they have been largely employed in text categorization by topic and other text classification tasks, like spam filtering [32] and sentiment analysis [92], whose bipartite nature (in addition to reasons reported above) makes them good targets for SVMs.

Several improvements have been proposed on support vector machines for text categorization: examples are the use of a kernel function based on external semantic knowledge [110], an *active learning* approach where users are only required to manually label few documents from an unlabeled set [115] and a *co-training* approach where two classifiers are iteratively trained on disjoint sets of features to integrate initially unlabeled documents [58].

3.5.3 Other methods

In the following are summarized some other relevant machine learning methods which have been proposed for text categorization.

- ***k*-nearest neighbors (*k*-NN)** classification consists into comparing each document to be classified with all those of the training set, finding the k most similar to it according to a defined similarity measure (e.g. cosine similarity) and labeling the new document with the most recurring category among these k training documents. This approach, in its basic form (without e.g. indexing documents for faster queries), is often defined *lazy*, as a labeled "training" set is needed but no specific operation is actually performed at training time. Heavy computation is instead deferred to the classification phase, which could take a relatively long time to compare each incoming document with the whole training set. Many works test the performances obtained with k-NN in comparison with other learners [123, 122]; other papers propose improvements to the basic approach, for example by assigning differentiated weights to terms [46] or training documents [113].

- ***Decision tree*** learners analyze training data to generalize it and infer a tree of simple rules, which are followed to classify new documents: the tree is explored top-down from the root and in each node

a branch is traversed according to the value of one feature, until a terminal node indicating the predicted class is reached. An interesting characteristic of decision tree models, compared to other types, is that they are easily interpretable by humans: for example, the discriminative feature used in the root node of a decision tree is usually among the ones giving the best clues about the class of each instance. Nowadays, the decision tree approach is not used for text categorization, but some early works proposed it [34, 67]. Decision trees are also employed in the method proposed in Chapter 5.

- ***Linear Least Squares Fit (LLSF)*** is in general a method to infer a linear function $f(\mathbf{d}) = \mathbf{Md}$, where $\mathbf{M}$ is the matrix for which, given a training set of pairs of vectors $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ and $\mathbf{Y} = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n)$, the error on it measured by the Frobenius norm $\|\mathbf{MI}-\mathbf{O}\|_N$ is minimized. Rather than classification, this method performs *regression*, meaning that one or more real values are predicted rather than one class or a set thereof. However, it has been applied to text categorization, using bags of words as input vectors and vectors with likelihood scores for each possible category as values, although also in this case has been proposed mostly in earlier works [122, 125].

3.6 Nearest centroid classification

Approaches to text categorization considered up to now rely on standard machine learning methods, commonly used also in non-text related tasks. Special focus is given here to the *Rocchio classifier* or *nearest centroid classifier* for having instead its roots mainly in information retrieval and especially for introducing the idea of explicit document-like representations of categories, which is prominently used in the methods presented in the next chapters.

Originally, the *Rocchio algorithm* is a method for relevance feedback in information retrieval. Relevance feedback is used to transform an arbitrary user-given search query on a set of documents basing on the results it gives initially to yield more accurate results.

Assume to have a set of documents $\mathcal{D} = \{\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_{|\mathcal{D}|}\}$, represented as vectors in a n-dimensional space, according to the bag-of-words model. A generic query given by a user would be likely mapped to a vector $\mathbf{q}$, which

is compared to all documents to retrieve a subset $\mathcal{D}_\mathbf{q} \subseteq \mathcal{D}$ of documents considered relevant to that query. At this point, in the Rocchio method, the following formula is used to compute a modified query vector $\mathbf{q}_m$, which is then used like the original vector $\mathbf{q}$ to retrieve a set of relevant documents.

$$\mathbf{q}_m = (\alpha \cdot \mathbf{q}) + \left(\beta \cdot \frac{1}{|\mathcal{D}_\mathbf{q}|} \sum_{\mathbf{d} \in \mathcal{D}_\mathbf{q}} \mathbf{d} \right) - \left(\gamma \cdot \frac{1}{|\mathcal{D} - \mathcal{D}_\mathbf{q}|} \sum_{\mathbf{d} \in \mathcal{D} - \mathcal{D}_\mathbf{q}} \mathbf{d} \right)$$

In practice, the modified query is computed from the sum of the following three factors, weighted according to the parameters α, β and γ:

- the original query,
- the mean of related documents,
- the mean of non-related documents (in negative).

This approach, intuitively, moves the query vector towards related documents and away from unrelated ones, in a measure dependent from the three weighting parameters. Specifically, what β and γ weight are the *centroids* of the sets $\mathcal{D}_\mathbf{q}$ and $\mathcal{D} - \mathcal{D}_\mathbf{q}$, which are used as representations of the whole sets.

In some works, this approach has been adapted to text categorization. Suppose that is given a set $\mathcal{D}_T$ of training documents organized in a set of categories $\mathcal{C}$ according to a labeling $\mathcal{L}$; from each document $d \in \mathcal{D}_T$ a vector $\mathbf{w}_d$ has been extracted. For each category $c \in \mathcal{C}$, having a set $\mathcal{D}_c = \{d \in \mathcal{D}_T : \mathcal{L}(d, c) = 1\}$ of training documents labeled with it, a representative vector $\mathbf{w}_c$ can be computed using the Rocchio's formula without considering the query factor.

$$\mathbf{w}_c = \left(\beta \cdot \frac{1}{|\mathcal{D}_c|} \sum_{d \in \mathcal{D}_c} \mathbf{w}_d \right) - \left(\gamma \cdot \frac{1}{|\mathcal{D} - \mathcal{D}_c|} \sum_{d \in \mathcal{D} - \mathcal{D}_c} \mathbf{w}_d \right)$$

As above, the parameters β and γ control respectively how much positive weight give to documents in c and how much penalty give to other documents. Usually, β is greater than γ and in some cases γ is 0, meaning that only training documents labeled with c contribute to build its vector.

Each vector $\mathbf{w}_c$ effectively serves as a representation (or *profile*) of a category c having the same form used to represent document: for this, such representations are sometimes referred to as *prototype documents*. Being in the same space, vectors for documents and categories can have their distance measured through standard methods. Once a suitable similarity function $s : \mathbb{R}^n \times \mathbb{R}^n \to [0, 1]$ in the vector space is defined, classification of any document d can be simply seen as finding the category $\hat{l}(d) = c$ having the representation $\mathbf{w}_c$ least distant from the vector $\mathbf{w}_d$ representing the document. This general approach is known as *nearest centroid classification*.

$$\hat{l}(d) = \underset{c \in \mathcal{C}}{\operatorname{argmax}}\, s(\mathbf{w}_d, \mathbf{w}_c)$$

The function most commonly used for this is cosine similarity, which as cited in §2.2.1 is particularly indicated to compare bags of words.

In the case of multi-label classification, a suitable threshold should be found so that a document is assigned to all categories having their profile at a distance lower than the threshold.

Likely to other approaches cited above, nearest centroid classification has been proposed in early works about machine learning-based text categorization [34], but experiments shown that it usually does not perform as good as methods like SVMs and Naïve Bayes. This weakness can be ascribed to the impossibility in many cases of accurately representing a category with a single centroid [122]. Anyway, given their straightforward structure and the efficiency in extracting them, we investigate the use of centroids in the methods presented in the following chapters, which are based on these aspects.

3.7 Hierarchical classification

As discussed in §3.2.2, in hierarchical classification problems, categories are organized in a rooted tree-like taxonomy, where categories farther from the root represent more specific topics with respect to higher-level categories.

In many practical cases, accurate classification of documents in a hierarchy of categories is difficult due to the usually very high number of categories with respect to typical flat categorization cases. Anyway, the knowledge of how categories are organized in the taxonomy can be lever-

aged to decompose the classification problem in order to address it with more efficacy and efficiency.

There are two main approaches to take this knowledge into consideration: conditioning the training of a single classifier operating on all categories of the taxonomy or building multiple classifiers for different nodes of the tree.

3.7.1 Big-bang approach

In the *big-bang* (or *global classifier*) approach, a single classification model is trained an all categories of the taxonomy, which is however taken into consideration in some way. This can be done either intervening in the feature extraction phase or using an appropriate learning algorithm able to cope with classes in a hierarchy.

Using the first method, in [14] is investigated the use of concept-based document representations to supplement word- or phrase-based features. Weak hypotheses are created based on terms and concepts features, which are combined using Adaboost. The method in [116] transform the classification output into a vector with boolean components corresponding to the possible category. They also use a distance-based metric to measure the similarity of training examples in the classification tree.

For what regards adapted learning algorithms, in [15] is proposed a generalization of support vector machines learning, where discriminant functions (used to classify instances) can be adapted to mirror structures such as a categories hierarchy. Multi-class SVMs are also used in [100], where in the computation of the optimal separation hyperplanes leaf categories are taken into consideration together with respective ancestor categories.

3.7.2 Local classifiers

Approaches with *local classifiers* are generally based on standard learning algorithms, but use them to train multiple knowledge models on different portions of the taxonomy. Specifically, a specialized classifier is trained on each intermediate node of the tree, possibly using a local set of predictive features. Classification of a document follows then a top-down process starting from the root of the taxonomy, where the classifier of each node predicts the correct branch for the document and, if no definitive category label is decided, forwards it to another more specific classifier. If documents

can be classified in non-leaf nodes, it is necessary to decide at each step whether to classify document at current node or keep walking down the tree.

In [33] SVM classifiers are used in a two-levels hierarchy of summaries of web documents, using feature sets created with documents and categories of the same node, assigning multiple leaf nodes to the test documents; similarly [75] also use multiple SVMs trained for each category. In [19] is proposed an incremental classifier, while [18] presents a refined evaluation scheme. In [121] is discussed a strategy based on pruning the original hierarchy which first computes the similarity between the test document and all other documents, and then classifies the document in a pruned hierarchy.

In [112] are tested three methods to limit the blocking problem in top-down approaches, that is the problem of documents wrongly rejected by the classifiers at higher-levels of the taxonomy. The *expert refinements* technique [4] uses cross-validation in the training phase to obtain a better estimation of the true probabilities of the predicted categories. In [104] is proposed a variant of the Hierarchical Mixture of Experts model, making use of two types of neural networks for intermediate nodes and for leaves.

A comparison between the usage of a standard flat classifier and local classifiers in a hierarchy is given in [17], testing both SVM and naïve Bayes approaches.

3.8 Experimental evaluation

As commonly happens in machine learning and data mining problems, while developing an improved or novel method for text classification, it is necessary to assess its effectiveness. Other than being theoretically motivated, every proposed method typically undergoes an *experimental evaluation* phase, where its accuracy is measured by applying it in a controlled environment where errors can be detected.

The typical experimental evaluation of a text classification method can be broken down to the following steps.

1. First, one or more *benchmark datasets* must be collected, each consisting of a significant collection $\mathcal{D}$ of text documents which already are coherently labeled with meaningful categories of a set $\mathcal{C}$, according to a labeling $\mathcal{L} : \mathcal{D} \times \mathcal{C} \to \{0, 1\}$. Throughout the literature, a

number of recurring datasets have been used as benchmark: these are presented below.

2. From the set of documents $\mathcal{D}$ of each of the datasets must be sampled a *training set* $\mathcal{D}_T \subseteq \mathcal{D}$ and a *test set* (or *evaluation set*) $\mathcal{D}_E \subseteq \mathcal{D}$. In order to correctly assess the general efficacy of the classifier, the two sets must be disjoint ($\mathcal{D}_T \cap \mathcal{D}_E = \emptyset$).

3. According to the specific method under analysis, the training set $\mathcal{D}_T$ is used to build the classifier(s).

4. This classifier is applied to all documents in $\mathcal{D}_E$: a predicted labeling $\hat{\mathcal{L}} : \mathcal{D}_E \times \mathcal{C} \to \{0, 1\}$ is obtained for these documents.

5. The predicted labeling $\hat{\mathcal{L}}$ is compared against the actual one $\mathcal{L}$ for the test documents. One or more accuracy measures are computed, giving a quantifiable assessment of the goodness of the classifier.

In step 2, to create a training and a test set from a set of documents, this is usually split into two parts, not necessarily of the same size: this is the basic *hold-out* method. Another used approach is *cross-fold validation*: the whole set of documents $\mathcal{D}$ is exhaustively partitioned into k disjoint sets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$, evaluation (steps 3-5) is then performed once for each set $\mathcal{D}_i$ using it as a test set and the complementary set $\mathcal{D} - \mathcal{D}_i$ for training; results for each i are then aggregated.

The goal of experimental evaluation is to verify how much the labeling $\hat{\mathcal{L}}$ predicted by a trained classifier for test documents $\mathcal{D}_E$ is similar to the actual labeling $\mathcal{L}$ known from the collection. There are multiple methods commonly used in machine learning and data mining to quantify this similarity: those to be used generally depend from the specific problem under analysis and there is no one which is universally acknowledged as the best one. Anyway, also for this aspect, there are some common choices, which will be discussed briefly.

3.8.1 Benchmark datasets

In order to perform experimental evaluation of a text classification method, at least one benchmark dataset must be retrieved; however, experimental results for a method are usually given for two or three of such datasets.

Throughout the literature, some collections of pre-classified documents, also known as *corpora* (singular: *corpus*), have been usually employed as benchmark datasets in text classification and some other text mining tasks. Using the same benchmark datasets across different works helps in making comparisons between the efficacy of each method, as the experimental results are measured with methods working in the same conditions. Should be noted, anyway, that sometimes different works use different parts of the same dataset, or perform a different split between training and test documents.

Here are presented some of these corpora commonly used as benchmarks. For some of them, one or more predefined splits between training and test documents exist.

- The ***Reuters-21578*** collection[2] is a corpus of newswire stories of 1987 about economy and markets, primarily edited by David Lewis; it is a cleaned up version of the older *Reuters-22178* collection. As the name suggests, it includes 21,578 documents. Different splits between training and test set have been used, making sometimes difficult the comparison of different works using this collection. One of the most common splits is *ModApté*: it contains 9,603 documents for training and 3,299 documents for testing (8,676 documents remain unused).

  To each document is assigned a subset of the possible labels, divided into 5 groups: TOPICS, ORGS, PEOPLE, PLACES and EXCHANGES: those of the first group are often used as categories. Documents are distributed quite unevenly between topics: documents counts for topics range from a handful to hundreds. Most works based on the ModApté split perform experiments considering one or both of two subsets of categories: the 90 topics having at least one document in both training and test set and the 10 topics with the greatest numbers of documents (more than 200 each in the union between training and test set).

- The ***20 Newsgroups*** collection[3] contains almost 20,000 newsgroups posts from Usenet discussion groups, partitioned nearly evenly across 20 different groups (about 1,000 posts per group). These 20 groups represent more or less specific topics of different areas, so that there

[2] `http://www.daviddlewis.com/resources/testcollections/reuters21578/`
[3] `http://qwone.com/~jason/20Newsgroups/`

are categories nearly related to each other (such as COMP.SYS.IBM.PC-.HARDWARE and COMP.SYS.MAC.HARDWARE), slightly more loosely related categories (TALK.RELIGION.MISC and ALT.ATHEISM) and highly unrelated categories (REC.AUTOS and TALK.POLITICS.GUNS). Also of this collection slight variants exist: a common one is the "bydate" selection of 18,846 posts, divided between 11,314 training posts and 7,532 test posts.

- ***Ohsumed*** is a subset of the MEDLINE database created by William Hersh and colleagues at the Oregon Health Sciences University and consists of 348,566 documents obtained from 270 medical journals over the period 1987 to 1991. The documents were manually indexed using the taxonomy MeSH (Medical Subject Headings), which consists of 18,000 categories. The choice used in the literature is the consideration of only the articles in the HEALTHDISEASES category and its subtree in the MeSH taxonomy. This sub-hierarchy has 119 categories, of which only 102 contain usable data.

- ***Reuters Corpus Volume 1 (RCV1)*** [68] is, like Reuters-21578, a collection of newswire stories, labeled with different types of codes, which in this case are topic, industry and region. The whole collection contains over 800,000 articles, so that only portions of it are sometimes used. The 103 topic codes, usually treated as categories, are organized in a hierarchy with 4 top categories (CORPORATE/INDUSTRIAL, ECONOMICS, GOVERNMENT/SOCIAL and MARKETS).

Apart from these collections built on purpose, some sources exist across the Web from where collections of documents used by some works have been extracted.

The ***Open Directory Project (DMOZ)*** (already cited in 3.2.2) is a collaboratively built web directory, organizing web pages into a detailed taxonomy of topics. As of writing this, it reports containing 4,129,130 links to web sites organized under more than 1,023,636 categories. While each category (apart from TOP) has a single "hard" parent node in the database, other nodes may have a link to it as a cross-related sub-topic: considering these links, the structure of the taxonomy has the form of a directed acyclic graph (DAG) rather than a tree (where only one path must exist between

Actual category	Predicted category			
	COMPUTER	MUSIC	SPORTS	*(total)*
COMPUTER	**532**	42	13	587
MUSIC	23	**547**	16	586
SPORTS	14	18	**563**	595
(total)	569	607	592	1768

Figure 3.3 – *Example of confusion matrix with three categories; entries in bold constitute the diagonal*

two nodes). ***Yahoo! Directory***[4] was another similarly structured web directory.

3.8.2 Evaluation metrics

In the following, specific methods to numerically evaluate classification accuracy from test runs of a method are presented.

In the case of single-label classification (including binary classification), for each test document $d \in \mathcal{D}_E$, there is one single actual class $l(d) \in \mathcal{C}$ and one single predicted class $\hat{l}(d) \in \mathcal{C}$: the document is correctly classified if and only if the two are equal. In machine learning, it is common to summarize the co-occurrences between actual and predicted classes in a *confusion matrix*, showing the number of documents for each possible couple of them: Figure 3.3 shows an example. The numbers in the main diagonal are counts of cases where actual and predicted labels match, so they refer to correctly classified documents; the rest of the values out of the diagonal represent various types of errors.

From the confusion matrix many performance measures can be computed. The most obvious one is the ratio of correctly classified documents with respect to the total, which is often referred to as *accuracy*: it is in practice the sum of the elements of the diagonal divided by the sum of all elements (i.e. the total number of test documents). For example, from the confusion matrix reported in Figure 3.3, an accuracy of $(532 + 547 + 563)/1768 \approx 0.9287 = 92.87\%$ can be computed.

[4] `http://dir.yahoo.com` (no longer active)

$$\mathsf{Acc} = \frac{|d \in \mathcal{D}_E : l(d) = \hat{l}(d)|}{|\mathcal{D}_E|}$$

The accuracy is generally used when there are few categories (ideally two) and documents are equally distributed among them. In a case where categories are unbalanced, the accuracy may underestimate errors done on minority categories. For example, supposing a test set with 90% of documents of class A and 10% of class B, even a trivial classifier indicating class A for any document would obtain a 90% accuracy, which is generally an high value.

Measures exist to evaluate performances on single categories. The *recall* $\rho(c)$ of a category c is the ratio of correct classifications within documents which are *actually* labeled with c, while its *precision* $\pi(c)$ is conversely the number of correct classifications within documents for which c has been *predicted* as category.

$$\rho(c) = \frac{|d \in \mathcal{D}_E : l(d) = \hat{l}(d) = c|}{|d \in \mathcal{D}_E : l(d) = c|} \qquad \pi(c) = \frac{|d \in \mathcal{D}_E : l(d) = \hat{l}(d) = c|}{|d \in \mathcal{D}_E : \hat{l}(d) = c|}$$

The two values measure in practice how much the classifier is able to detect category c and to assign it only to correct documents. They are usually summarized in a *F-measure*, parameterized by a factor β which indicates the importance to give to recall with respect to precision

$$F_\beta(c) = \frac{(\beta^2 + 1) \cdot \rho(c) \cdot \pi(c)}{\beta^2 \cdot \rho(c) + \pi(c)}$$

The most common choice is to set $\beta = 1$, i.e. equally weighting recall and precision: this results in their harmonic mean, known as F_1 *measure*.

$$F_1(c) = \frac{2 \cdot \rho(c) \cdot \pi(c)}{\rho(c) + \pi(c)}$$

Consider now multi-label classification, where each test document $d \in \mathcal{D}_E$ has a set $L(d) = \{c \in \mathcal{C} : \mathcal{L}(d, c) = 1\}$ of actual labels and a set $\hat{L}(d) = \{c \in \mathcal{C} : \hat{\mathcal{L}}(d, c) = 1\}$ of predicted labels, rather than only one of both. The classifier in this case can be seen as taking $|\mathcal{D}_E \times \mathcal{C}|$ decisions between c (label the document with the category) and $\overline{c}$ (do not label)

Actual class	**Predicted class**	
	c	$\overline{c}$
c	TP_c (*true positives*)	FN_c (*false negatives*)
$\overline{c}$	FP_c (*false positives*)	TN_c (*true negatives*)

Figure 3.4 – *Scheme of a confusion matrix for a single category c*

rather than $|\mathcal{D}_E|$ decisions among categories in $\mathcal{C}$.

In this case, the standard accuracy cannot be computed as above, as each document may have an arbitrary number of actual and predicted categories. One obvious solution would be to compute the ratio of test documents for which the classifier predicted its exact set of categories, but this ratio would be generally quite small and can easily underestimate the efficacy of the classifier. On the other hand, the accuracy could be computed as the ratio of correct decisions of the classifier on single couples $(d, c) \in \mathcal{D}_E \times \mathcal{C}$, but this tends instead to overestimate the efficacy. This is due to the fact that usually each document belongs to a small ratio of all the possible categories, so there is a vast majority of decisions which should be negative and usually most of them are trivial, as documents are very easily detected as unrelated to most categories.

In multi-label evaluation, it is not possible to extract a single confusion matrix of all categories. Instead, considering an independent binary classification problem for each category, one binary confusion matrix for each category c can be consequently extracted, having c and its complement $\overline{c}$ as classes. Each of these matrices (also known as *tables of confusion*) show four counts related to a category c:

true positives documents correctly labeled with c,

false negatives documents for which the c label is erroneously missing,

false positives documents wrongly labeled with c,

true negatives documents correctly not labeled with c.

Recall $\rho(c)$ and precision $\pi(c)$ for each category c can be computed likely as above, adapting the formulas to the multi-label case. From the two, F_1 measure is computed exactly as above.

$$\rho(c) = \frac{TP_c}{TP_c + FN_c} = \frac{|d \in \mathcal{D}_E : c \in L(d) \cap \hat{L}(d)|}{|d \in \mathcal{D}_E : c \in L(d)|} = \frac{\sum_{d \in \mathcal{D}_E} \mathcal{L}(d,c) \cdot \hat{\mathcal{L}}(d,c)}{\sum_{d \in \mathcal{D}_E} \mathcal{L}(d,c)}$$

$$\pi(c) = \frac{TP_c}{TP_c + FP_c} = \frac{|d \in \mathcal{D}_E : c \in L(d) \cap \hat{L}(d)|}{|d \in \mathcal{D}_E : c \in \hat{L}(d)|} = \frac{\sum_{d \in \mathcal{D}_E} \mathcal{L}(d,c) \cdot \hat{\mathcal{L}}(d,c)}{\sum_{d \in \mathcal{D}_E} \hat{\mathcal{L}}(d,c)}$$

In order to get an overall evaluation of the classifier, these values must be aggregated. Two approaches exist to compute an average of recall and precision for all categories. The *macroaverage* recall (or precision) is simply the average of recalls (or precisions) for all categories. Conversely, *microaverage* recall and precision are computed from the confusion matrix obtained by summing up by components those for each category: they refer in practice to all the individual decisions for each couple $(d, c) \in \mathcal{D}_E \times \mathcal{C}$.

$$\rho^m = \frac{\sum_{c \in \mathcal{C}} TP_c}{\sum_{c \in \mathcal{C}} TP_c + \sum_{c \in \mathcal{C}} FN_c} \qquad \pi^m = \frac{\sum_{c \in \mathcal{C}} TP_c}{\sum_{c \in \mathcal{C}} TP_c + \sum_{c \in \mathcal{C}} FP_c}$$

All these accuracy measures, as stated above, can be applied in the case of an *hard* binary decision in $\{0, 1\}$ for each document-category couple. Anyway, some classifiers commonly used in text classification (including some of those presented in Section 3.5) return by default a likelihood $p(d, c) \in [0, 1]$ of any document d belonging to any category c which may be equal to 0, 1 or any intermediate value between them. In this case, hard decisions can be obtained by setting a threshold value τ so that a positive decision is taken for all couples and only for those for which the likelihood score is greater (or equal) to the threshold.

By varying the value of the threshold, some different classification decisions are taken about test documents. More specifically, if the threshold value is low, documents will be generally labeled with more categories and the number of missing correct labels will decrease (high recall), but documents will be more likely be filed under wrong categories (low precision). Conversely, if the threshold is set to a high value, documents will more hardly be labeled with categories and wrong labels will drop (high precision), but more correct labels will be also missed (low recall).

Summing up, the threshold can be set accordingly to the desired tradeoff

between recall (avoiding missing labels) and precision (avoiding erroneous labels): as the threshold grows, recall drops and precision increases. In many past works, was common to search the threshold setting for which precision and recall are equal (or as close as possible): their common value with this setting is called *breakeven point* (BEP) and was often used as an overall accuracy measure with a settable threshold.

For what concerns hierarchical classification, with reference to a single-label case, the accuracy can be computed as above as the ratio of correctly classified test documents. However, distinctions can be made on the errors made by the classifier: the document may be assigned to an overly generic or specific category with respect to the correct one, or may be even classified under a category of a different branch of the tree, which can be at different distances from the correct one.

When evaluating the classifier, one could want to weight these errors according to their gravity: for example, intuitively, classifying a document in a sibling (i.e. a node with the same parent) of the correct category is a smaller error than choosing the wrong path in the tree already at the root, thus ending in a very distant category. For example, with reference to the taxonomy in Figure 3.1 (page 39), classifying a document about DRAMA under POETRY could be deemed as a more "forgivable" mistake than classifying it under MOVIES or even worse under MANAGEMENT.

Some different measures for evaluation of hierarchical classification have been proposed. Among these, *hierarchical recall and precision* have been presented in [59] and endorsed by [109] as an intuitive adaptation of the standard recall and precision concepts to this context. Considering that each document classified under a category c is also automatically labeled with all ancestor categories, the hierarchical recall $\rho^H(d)$ is computed for each document d as the ratio of actual labels $L(d)$ which have been predicted, while hierarchical precision $\pi^H(d)$ is the ratio of predicted labels $\hat{L}(d)$ which are correct. In all cases, the root node is not considered.

- For a correct classification both recall and precision are 1.
- For a *generalization* error (a document assigned in a correct but too generic category) the precision is 1 and the recall is proportional to the number of correct labels found.
- For a specialization error (a document labeled with too specific classes) the recall is 1 and the precision decreases as the predicted category is

more deep in the taxonomy.

- For other errors, recall and precision are both below 1 and are penalized respectively by the unpredicted actual categories and the erroneously assigned categories.

For example, still referencing the taxonomy in Figure 3.1, for a document d having DRAMA as its correct category, classifying it generically under ARTS would imply a recall $\rho^H(d) = \frac{1}{3}$ (as LITERATURE and DRAMA are missed) and a precision $\pi^H(d) = 1$ (as ARTS is correct); classifying it under MOVIES would instead yield $\rho^H(d) = \frac{1}{3}$ and $\pi^H(d) = \frac{1}{2}$.

For multiple test documents, the measures are microaveraged: the result is the ratio of the sums of the two ratio elements for each document.

$$\rho^H = \frac{\sum_{d \in \mathcal{D}_E} |L(d) \cap \hat{L}(d)|}{\sum_{d \in \mathcal{D}_E} |L(d)|} \qquad \pi^H = \frac{\sum_{d \in \mathcal{D}_E} |L(d) \cap \hat{L}(d)|}{\sum_{d \in \mathcal{D}_E} |\hat{L}(d)|}$$

From the two, a *hierarchical* F_1 *measure* can be computed in the usual way.

$$F_1^H = \frac{2\rho^H \pi^H}{\rho^H + \pi^H}$$

Chapter 4

Cross-Domain Text Categorization

This chapter deals with the specific task of *cross-domain* text categorization, where knowledge extracted from labeled documents must be applied to other documents of a different domain where different terms are used, so that methods described so far do not achieve optimal results. After an introduction of the problem, known methods to accomplish this task are exposed, then one novel solution to it along with experimental results is presented.

4.1 Problem description

As seen in the previous chapter, typical approaches to automatic text classification in predefined categories require the availability of a training set of documents, from which necessary knowledge is inferred. A training set generally must have some characteristics in order to be usable and to yield optimal results: it must contain an adequate amount of documents, which must be as exactly as possible representative of the documents which will be classified and must be coherently labeled with the same categories which are going to be assigned to new documents. To be representative, training documents should intuitively contain the same words or, more generally, the same features which will be extracted to predict categories of subsequent documents to be classified, which are refered to as *target* documents.

In other words, can be said that training documents must be within the same *domain* of the target documents. Intuitively, the domain of the documents is the context within they are produced and/or consumed and dictates the words which are used and the categories under which are or must be filed.

Retrieving a set of documents of the exact same domain of the target documents can often be difficult. As discussed previously, a solution would be to manually label some target documents, but creating a suitable training set may require to label a great amount of documents, thus implying a significant amount of human effort.

However, in some cases, a set of labeled documents of a slightly different domain may be available. The difference between the domains may consist in the use of some different terms or in the organization within categories representing slightly different concepts. Considering the general setting of the problem, the traditional techniques seen so far for text categorization might be applied to infer a classifier from available training documents and apply it to target documents. However, due to the differences between the two domains, this would likely not result in an accurate classification, as many of the features known by the classifier would not be found within the target documents.

Cases like these would require specific methods to somehow transfer the knowledge extracted from the available training data to the target domain. Throughout the last decade, techniques for *transfer learning* have been devised to address these cases. Transfer learning generally involves solving a problem in a *target domain* by leveraging knowledge from a *source domain* whose data is fully available. *Cross-domain text categorization* refers to the specific task of classifying a set of target documents in predefined categories using as support a set of pre-labeled documents of a slightly different domain.

Contrarily to traditional text categorization problems, where target documents are generally assumed to be not known in advance, typical cross-domain methods imply that unlabeled target documents must be given in advance, at least in part, as some knowledge of the target domain is necessary. Also, the majority of cross-domain methods consider a single-label setting (i.e. one and only one category label to each document), which is assumed by default for the rest of this chapter.

4.1.1 Formalization

Formally, an algorithm for cross-domain text classification has as its input a set $\mathcal{D}_S$ of *source documents* constituting the *source domain*, a set $\mathcal{D}_T$ of *target documents* making up the *target domain*, a set $\mathcal{C}$ of *categories* and a labeling $C_S : \mathcal{D}_S \to \mathcal{C}$ associating a single category to each source document. The required output is a predicted labeling $\hat{C}_T : \mathcal{D}_T \to \mathcal{C}$ for documents of the target domain.

For what concerns the relationship between the two domains, in the general case of transductive transfer learning where cross-domain text categorization falls in (see below), they must share the same feature space $\mathcal{X}$ and the same class labels $\mathcal{Y}$: in the case of text documents, the first condition can be satisfied simply by selecting the same terms for source and target domain. The common assumption on the difference between the two domains is that the labels are equally conditioned by the input data, which though is distributed differently in them. Denoting with X_S and Y_S data and labels for the source domain and with X_T and Y_T those for the target domain, we have $P(Y_S|X_S) = P(Y_T|X_S)$, but $P(X_S) \neq P(X_T)$: this condition is known as *covariate shift* [108].

4.1.2 Motivations

Cross-domain text categorization methods generally address situations where a set of target documents must be classified and a set of labeled documents of a slightly different domain is available.

A situation of this kind can be the classification of documents in a set of categories $\mathcal{C}_T$ by leveraging document pre-labeled with categories of a set $\mathcal{C}_S$ dealing with similar topics, such that there is a one-to-one mapping between $\mathcal{C}_S$ and $\mathcal{C}_T$. Consider for example, drawing category labels from the taxonomy depicted in Figure 3.1 (page 39), to have unlabeled documents about either MUSIC or PHYSICS to be distinguished from each other: if one happens to have pre-labeled documents about MOVIES and MATH, cross-domain methods can be used to extract knowledge from them which is then transferred to the similar categories. This problem can be seen as classification of documents in two general categories ARTS and SCIENCE, each including one sub-category of the source domain and one of the target. Datasets typically used as benchmarks for evaluation simulate this situation (§4.3.1).

An interesting area of application of cross-domain text categorization on which current research is focused is sentiment analysis, specifically the classification of texts (forum posts, reviews, etc.) talking about a specific entity as either positive or negative. Through cross-domain methods, it is potentially possible to perform this classification on reviews of objects of one type by training a classifier on reviews of different objects: it is possible for example to classify comments about books by training on reviews of movies which are already labeled for their polarity (e.g. with a 1-to-5-stars rating), or similarly to distinguish positive and negative reviews of hotels by learning from those addressed to restaurants. As for the other cases, cross-domain learning works better if the two domains are enough similar and there are many common words between the two.

4.2 State of the art

As stated above, cross-domain text categorization lies within the wide research branch of *transfer learning*, generally dealing with methods to transfer knowledge from a source to a target domain. A comprehensive survey on transfer learning is given in [90], where different types of tasks are described. In this survey, general cross-domain classification of data is referred to as *transductive transfer learning*, roughly meaning that the domains are different, but the classes are the same[1]. Text categorization is perhaps the specific task most commonly addressed within transductive methods: some works presented in the following are specific for this, while other are general approaches or can at least be easily adapted to other tasks.

Two major approaches to transductive transfer learning are generally distinguished: *instance transfer* and *feature representation transfer*, treated separately in the following.

4.2.1 Instance transfer

Instance transfer-based approaches generally work by re-weighting instances (data samples) from the source domain to adapt them to the target domain, in order to compensate the discrepancy between $P(X_S)$ and $P(X_T)$: this

[1] As the survey points out, this is different from the usual meaning of "transductive" in machine learning, used instead to indicate learning algorithms where test data must be known at training time.

generally involves estimating an *importance* $\frac{P(x_S)}{P(x_T)}$ for each source instance x_S to reuse it as a training instance x_T under the target domain.

Some works mainly address the related problem of sample selection bias, where a classifier must be learned from a training set with a biased data distribution. [128] analyzes the bias impact on various learning methods and proposes a correction method using knowledge of selection probabilities.

The *kernel mean matching* method [50] learns re-weighting factors by matching the means between the domains data in a reproducing kernel Hilbert space (RKHS); this is done without estimating $P(X_S)$ and $P(X_T)$ from a possibly limited quantity of samples. Among other works operating under this restriction there is the *Kullback-Liebler importance estimation procedure* [111], a model to estimate importance based on minimization of the Kullback-Liebler divergence between real and expected $P(X_T)$.

Among works specifically considering text classification, [25] trains a Naïve Bayes classifier on the source domain and transfers it to the target domain through an iterative Expectation-Maximization algorithm. In [42] multiple classifiers are trained on possibly multiple source domains and combined in a *locally weighted ensemble* based on similarity to a clustering of the target documents to classify them.

4.2.2 Feature representation transfer

Approaches based on feature representation transfer generally work by finding a new feature space to represent instances of both source and target domains, where their differences are reduced and standard learning methods can be applied.

The *structural correspondence learning* method [6] works by introducing *pivot* features and learning linear predictors for them, whose resulting weights are transformed through Singular Value Decomposition and then used to augment training data instances. The paper [28] presents a simple method based on augmenting instances with features differentiating source and target domains, possibly improvable through nonlinear kernel mapping. In [72] a spectral classification-based framework is introduced, using an objective function which balances the source domain supervision and the target domain structure. With the *Maximum Mean Discrepancy (MMD) Embedding* method [87], source and target instances are brought to a common low-dimensional space where differences between data distributions are

reduced; *transfer component analysis* [89] improves this approach in terms of efficiency and generalization to unseen target data.

The following works are focused on text classification. In [24] an approach based on co-clustering of words and documents is used, where labels are transferred across domain using word clusters as a bridge. The *topic-bridged PLSA* method [120] is instead based on Probabilistic Latent Semantic Analysis, which is extended to accept unlabeled data. In [129] is proposed a framework for joint non-negative matrix tri-factorization of both domains. *Topic correlation analysis* [69] extracts both shared and domain-specific latent features and groups them, to support higher distribution gaps between domains.

4.2.3 Other related works

Likely to traditional text classification, some methods leverage external knowledge bases: these can be helpful to link knowledge across domains. The method presented in [118] improves the cited co-clustering-based approach [24] by representing documents with concepts extracted from Wikipedia. The *bridging information gap* method [119] exploits instead an auxiliary domain acting as a bridge between source and target, using Wikipedia articles as a practical example. These methods usually offer very high performances, but need a suitable knowledge base for the context of the analyzed documents, which might not be easily available for overly specialized domains.

Other than text classification by topic, another related task on which domain adaptation is frequently used is *sentiment analysis*, where positive and negative opinions about specific objects (products, brands, etc.) must be distinguished: a usual motivating example is the need to extract knowledge from labeled reviews for some products to classify reviews for products of a different type, with possibly different terminologies. *Spectral feature alignment* [88] works by clustering together words specific for different domains leveraging the more general terms. In [8] a sentiment-sensitive thesaurus is built from possibly multiple source domains. In [20] a Naïve Bayes classifier on syntax trees-based features is used.

Beyond the presented works where domains differ only in the distribution of terms, methods for *cross-language* text classification exist, where source and target documents are written in different languages, so that there are few or no common words between the two domains. This scenario generally requires either some labeled documents for the target domain or

an external knowledge base to be available: a dictionary for translation of single terms is often used. As examples, in [73] is presented an approach based on *information bottleneck* where Chinese texts are translated into English to be classified, while the method in [99] is based on the structural correspondence learning cited above [6].

4.3 Evaluation

Throughout all works on cross-domain text categorization presented in the previous section, there is a small set of publicly available datasets used as benchmarks for experimental evaluation of them. All these datasets are generally organized in a shallow hierarchical structure, where a small number of *top-level categories* can be isolated so that each includes a number of roughly similar *sub-categories*.

An evaluation dataset can be thus set up by choosing a small set of top categories of a collection constituting the set $\mathcal{C}$ of possible categories and splitting documents of these categories into two groups: one contains documents of some branches of the top categories and is used as the source domain, the other one containing documents of different sub-categories is used as the target domain. By labeling each document in the two domains with its top-category, a dataset suitable for evaluation is obtained.

The tested method receives in input all the documents and the labeling of the source group and must give in output predicted labels for documents of the target group, which will be compared to known labels. Given the structure of the available datasets, the number of top categories is in most cases limited to two, with few works also performing tests with three or (rarely) four top categories, which usually give results sensibly worst compared to those on two categories only.

Given the single-label setting, the very small number of categories and in most cases the balanced distribution of documents across them, the basic *accuracy*, intended as the ratio of correctly classified documents with respect to the total, is mostly used as the unique performance indicator.

4.3.1 Common benchmark datasets

Here are presented the three recurring text collections for evaluation of topic-based cross-domain text categorization. The first two were already

presented in §3.8.1.

The ***20 Newsgroups*** collection is used for having its 20 categories divided among 6 main branches: the 4 most frequent of them are COMP, REC, SCI and TALK and are used as top categories, each represented by 4 sub-categories (5 for COMP). While many splits of sub-categories between source and target domain are possible, most works only consider one of them as a reference for each possible set of top categories. An example of splits for each of the six possible sets of two categories is given by [24] and reused in other subsequent works; this is reported in Table 4.1 and will be used as a reference in the following. For what regards the four possible sets of three categories and the full set of four categories, splits used as reference are taken from [118] (with the exception of COMP vs. REC vs. TALK, missing from the paper and made up here) and reported in Tables 4.2 and 4.3.

In the ***Reuters-21578*** collection, documents are labeled with 5 types of labels: while those of type TOPICS are often used as topic labels in regular text categorization tasks, ORGS, PEOPLE and PLACES are instead used as top categories for cross-domain classification. For each of the three possible pairs of top categories, sub-categories are usually evenly divided.

The ***SRAA*** text collection[2] is not much used for regular text categorization, but is very adapt for the cross-domain case. Like 20 Newsgroups, it is drawn from Usenet: it consists of 73,218 posts from discussion groups about simulated autos, simulated aviation, real autos and real aviation. With this setting, tests can be intuitively performed using two different sets of top categories: {REAL, SIMULATED} and {AUTO, AVIATION}. Given the highly unbalanced distribution of documents among the four categories in the base collection, tests are usually performed on a selection of 4,000 documents for each of them, for a total of 16,000 documents.

4.4 Iterative refining of category representations

In the following, a novel approach to cross-domain text categorization is proposed, summarily based on creating representations of categories from the source domain and adapting them to the target domain by iteratively

[2] `http://people.cs.umass.edu/~mccallum/data/sraa.tar.gz`

Table 4.1 – *Reference two-categories splits for cross-domain categorization experiments on the 20 Newsgroups datasets*

Top categories	Source domain	Target domain
COMP vs. SCI	**comp**.graphics	**comp**.sys.ibm.pc.hardware
	comp.os.ms-windows.misc	**comp**.sys.mac.hardware
		comp.windows.x
	sci.crypt	**sci**.med
	sci.electronics	**sci**.space
REC vs. TALK	**rec**.autos	**rec**.sport.baseball
	rec.motorcycles	**rec**.sport.hockey
	talk.politics.guns	**talk**.politics.mideast
	talk.politics.misc	**talk**.religion.misc
REC vs. SCI	**rec**.autos	**rec**.motorcycles
	rec.sport.baseball	**rec**.sport.hockey
	sci.med	**sci**.crypt
	sci.space	**sci**.electronics
SCI vs. TALK	**sci**.electronics	**sci**.crypt
	sci.med	**sci**.space
	talk.politics.misc	**talk**.politics.guns
	talk.religion.misc	**talk**.politics.mideast
COMP vs. REC	**comp**.graphics	**comp**.os.ms-windows.misc
	comp.sys.ibm.pc.hardware	**comp**.windows.x
	comp.sys.mac.hardware	
	rec.motorcycles	**rec**.autos
	rec.sport.hockey	**rec**.sport.baseball
COMP vs. TALK	**comp**.graphics	**comp**.os.ms-windows.misc
	comp.sys.mac.hardware	**comp**.sys.ibm.pc.hardware
	comp.windows.x	
	talk.politics.mideast	**talk**.politics.guns
	talk.religion.misc	**talk**.politics.misc

Table 4.2 – *Three- and four-categories splits used for cross-domain categorization experiments on the 20 Newsgroups collection*

Top categories	Source domain	Target domain
COMP vs. REC vs. SCI	**comp**.graphics	**comp**.os.ms-windows.misc
	comp.sys.ibm.pc.hardware	**comp**.windows.x
	comp.sys.mac.hardware	
	rec.motorcycles	**rec**.autos
	rec.sport.hockey	**rec**.sport.baseball
	sci.med	**sci**.crypt
	sci.space	**sci**.electronics
REC vs. SCI vs. TALK	**rec**.autos	**rec**.sport.baseball
	rec.motorcycles	**rec**.sport.hockey
	sci.med	**sci**.crypt
	sci.space	**sci**.electronics
	talk.politics.guns	**talk**.politics.mideast
	talk.politics.misc	**talk**.religion.misc
COMP vs. SCI vs. TALK	**comp**.graphics	**comp**.os.ms-windows.misc
	comp.sys.mac.hardware	**comp**.sys.ibm.pc.hardware
	comp.windows.x	
	sci.crypt	**sci**.space
	sci.electronics	**sci**.med
	talk.politics.mideast	**talk**.politics.misc
	talk.religion.misc	**talk**.politics.guns
COMP vs. REC vs. TALK	**comp**.graphics	**comp**.sys.ibm.pc.hardware
	comp.sys.mac.hardware	**comp**.windows.x
	comp.os.ms-windows.misc	
	rec.autos	**rec**.sport.baseball
	rec.motorcycles	**rec**.sport.hockey
	talk.politics.guns	**talk**.politics.misc
	talk.religion.misc	**talk**.politics.mideast

Table 4.3 – *Four-categories split used for cross-domain categorization experiments on the 20 Newsgroups collection*

Top categories	Source domain	Target domain
	comp.graphics	**comp**.sys.mac.hardware
	comp.os.ms-windows.misc	**comp**.sys.ibm.pc.hardware
		comp.windows.x
COMP VS. REC	**rec**.autos	**rec**.sport.baseball
VS. SCI VS.	**rec**.motorcycles	**rec**.sport.hockey
TALK	**sci**.crypt	**sci**.space
	sci.electronics	**sci**.med
	talk.politics.mideast	**talk**.politics.misc
	talk.religion.misc	**talk**.politics.guns

retrieving highly related documents and using them to refine the representations.

4.4.1 Rationale

Recalling the nearest centroid classifier (or *Rocchio classifier*) discussed in Section 3.6, classification of text documents can be carried out by producing prototypical representations of the known categories and finding for each document to be classified which is the most similar to it.

In the case of cross-domain categorization, if one had representations of the categories of the target domain, the nearest centroid approach could be used to accurately classify them. Anyway, these representations are not readily available, as they would require a correct labeling of the target documents, which is to be found.

What can be extracted from the available data of a cross-domain problem, instead, are centroids of categories in the source domain, as both documents and their labeling are known. As these categories are different from those of the target domain, representations generally differ from the optimal ones.

However, given the setting of the problem where categories of source and target domain are similar one-to-one, their representations are also expected to be similar. Specifically, considering the sets of words having

the highest weights in the representations, a number of these will be unique to a source category or to its correspondent in the target domain, but some of them will be shared between the two, thus boosting their similarity.

Considering this, if representations of categories extracted from source domain are compared to documents in the target domain, it is expected that nearest centroid classification will not be completely accurate, as the used profiles do not properly represent the domain. Anyway, a number of documents is likely to be associated to the respective correct categories. It is also expected that at least some of these documents happen to have a particularly high similarity to the prototype for their category, so that the confidence in their classification can be considered significantly high.

If all the assumptions hold, these documents are, with high probability, correctly classified within the target domain. From this set of labeled documents, new centroids of categories can be extracted as usual. Being extracted from a possibly small subset of the target documents, these representations are likely to constitute an approximation of categories of the target domain, but should be closer to the correct representations with respect to those originally extracted from the source domain.

In other words, it is expected that the newly obtained category profiles constitute an improvement over the previous ones obtained from source documents and thus, performing nearest centroid classification again using these, will yield more accurate predictions on the target documents. Documents which were used to create new representations will likely still be associated to them with even higher confidence and, additionally, new documents will potentially reach a high similarity to their category profile as before.

All these documents can be used to create a further new representation of categories, from which the same sequence of operations can be repeated. This cycle can be iterated an arbitrary number of times, with centroids of categories moving at each step, presumably getting close to the optimal representations of categories in the target domain.

This procedure is similar to the *k-means* clustering algorithm, where centers are progressively moved to centroids of respective data points until they converge to a stable configuration. Likely, the iterative algorithm proposed above can be stop when profiles of categories cease to change from one step to another. At this point, nearest centroid classification can be run one last time to get the definitive predicted category for each target document. If representations of categories successively moved close to real

centroids for the target domain, classification will be as accurate as possible.

This is the idea behind the proposed *iterative refining* method for cross-domain text categorization, formally described below.

4.4.2 Base method

The data to be given in input to the algorithm is what discussed in Section 4.1.1: the sets $\mathcal{D}_S$ and $\mathcal{D}_T$ of source and target documents whose union is denoted with $\mathcal{D}$, the set $\mathcal{C}$ of possible category labels shared across the domains and the labeling $C_S : \mathcal{D}_S \to \mathcal{C}$ mapping source documents to categories. In addition, the following two parameters are given:

- a *confidence threshold* $\rho \in [\frac{1}{|\mathcal{C}|}, 1)$ used to select relevant documents in the iterative phase,
- a maximum number N_I of iterations for the iterative phase.

The first step consists in typical document pre-processing, considering the whole set of all known documents $\mathcal{D} = \mathcal{D}_S \cup \mathcal{D}_T$. At this extent, the following steps are performed in detail for each document:

- words are extracted by splitting on all non-letter characters (spaces, periods, dashes, ...),
- each word is casefolded (all letters are turned lowercase) for uniformity,
- words shorter than 3 letters are removed,
- words appearing in a predefined list of 671 stopwords are removed[3],
- Porter's stemming algorithm is applied to each word.

After this, distinct words along with their counts of occurrences can be obtained for each document. As correct labels are known only for source documents and not for target, unsupervised techniques for term selection and weighting are employed. Selecting features on both domain generally ensures to not remove words appearing in only one of them. Specifically,

[3]The list was retrieved from `http://www.ranks.nl/resources/stopwords.html`; actually, some of the 671 words are already filtered out by previous steps.

feature selection is simply based on document frequency: terms appearing in less than 3 documents are discarded. For what concerns term weighting, various schemes will be tested. In the end, from each document $d \in \mathcal{D}$ a vector $\mathbf{w}_d$ of term weights is extracted.

From the bags of words for source documents, whose labeling is known, an initial representation $\mathbf{w}_c^0$ for each category $c \in \mathcal{C}$ can be computed as the centroids of relevant documents, whose set is denoted with R_c^0.

$$R_c^0 = \{d \in \mathcal{D}_S : C_S(d) = c\}$$

$$\mathbf{w}_c^0 = \frac{1}{|R_c^0|} \sum_{d \in R_c^0} \mathbf{w}_d$$

At this point, the iterative phase begins by measuring the similarity of these representations with bags of words for target documents by means of cosine similarity between the vectors, as discussed in §2.2.1. For each couple $(d, c) \in \mathcal{D}_T \times \mathcal{C}$, a relatedness score $s^0(d, c) = \cos(\mathbf{w}_d, \mathbf{w}_c^0)$ is thus obtained. To get for each document $d \in \mathcal{D}_T$ a distribution of probabilities across the possible categories, the score for each category is divided by the sum of those for all categories, so that they sum to one: the result for each category c is referred to as $p^0(d, c)$.

$$p^0(d, c) = \frac{s^0(d, c)}{\sum_{\gamma \in \mathcal{C}} s^0(d, \gamma)}$$

These *relative* probabilities are compared with the confidence threshold ρ to select a set $R_c^1 \subseteq \mathcal{D}_T$ of representative documents for each category $c \in \mathcal{C}$. Specifically, documents chosen for each category c are those whose relative probability both outscores those for other categories and is superior to the ρ threshold.

$$R_c^1 = \{d \in \mathcal{D}_T : p^0(d, c) > \rho \wedge p^0(d, c) \geq p^0(d, \gamma) \forall \gamma \in \mathcal{C}\}$$

Now, a new representation for each category can be computed as the centroid of these representative documents.

$$\mathbf{w}_c^1 = \frac{1}{|R_c^1|} \sum_{d \in R_c^1} \mathbf{w}_d$$

This completes the first iteration of the refining loop. For each subsequent iteration i, executed computations are obtained by generalizing the above formulas.

$$s^i(d,c) = \cos(\mathbf{w}_d, \mathbf{w}_c^i) \qquad p^i(d,c) = \frac{s^i(d,c)}{\sum_{\gamma \in \mathcal{C}} s^i(d,\gamma)}$$

$$R_c^{i+1} = \{d \in \mathcal{D}_T : p^i(d,c) > \rho \wedge p^i(d,c) \geq p^i(d,\gamma) \forall \gamma \in \mathcal{C}\}$$

$$\mathbf{w}_c^{i+1} = \frac{1}{|R_c^{i+1}|} \sum_{d \in R_c^{i+1}} \mathbf{w}_d$$

At the end of each iteration i, i.e. after computing the new category profiles, stop conditions are checked. The iterative phase terminates when

- the limit of N_I iterations is reached ($i + 1 = N_I$), *or*
- representations for all categories are identical to those obtained after the previous iteration.

$$\forall c \in \mathcal{C} : \mathbf{w}_c^{i+1} = \mathbf{w}_c^i$$

In this case, the algorithm terminates by measuring once again similarity between category profiles obtained at last iteration and predicting for each target document the category having the most similar profile.

$$\hat{C}_T(d) = \underset{c \in \mathcal{C}}{\operatorname{argmax}} \cos(\mathbf{w}_d, \mathbf{w}_c^N)$$

The pseudo-code for the whole algorithm (without considering the common documents pre-processing phase) is shown in Figure 4.1: as can be seen, the process is quite straightforward, allowing it to be easily implemented in many possible programming languages and environments.

4.4.3 Computational complexity

The process performs many operations on vectors of length $|\mathcal{T}|$: while these operations would generally require a time linear in this length, given the prevalent sparsity of these vectors, we can use suitable data structures to bound both storage space and computation time linearly w.r.t. the mean

Input: a bag of words $\mathbf{w}_d$ for each document $d \in \mathcal{D} = \mathcal{D}_S \cup \mathcal{D}_T$, set $\mathcal{C}$ of top categories, labeling $C_S : \mathcal{D}_S \to \mathcal{C}$ for source documents, confidence threshold ρ, maximum number N_I of iterations
Output: predicted labeling $\hat{C}_T$ for documents of the target domain

for all $c \in \mathcal{C}$ **do**
  $R_c^0 \leftarrow \{d \in \mathcal{D}_S : C_S(d) = c\}$
  $\mathbf{w}_c^0 \leftarrow \frac{1}{|R_c^0|} \cdot \sum_{d \in R_c^0} \mathbf{w}_d$
end for
$i \leftarrow 0$
while $i < N_I \wedge (i = 0 \vee \exists c \in \mathcal{C} : R_c^i \neq R_c^{i-1})$ **do**
  for all $(d, c) \in \mathcal{D}_T \times \mathcal{C}$ **do**
    $s^i \leftarrow \cos(\mathbf{w}_d, \mathbf{w}_c^i)$
    $p^i(d, c) \leftarrow \frac{s^i(d,c)}{\sum_{\gamma \in \mathcal{C}} s^i(d,\gamma)}$
  end for
  for all $c \in \mathcal{C}$ **do**
    $A_c^i \leftarrow \{d \in \mathcal{D}_T : \operatorname{argmax}_{\gamma \in \mathcal{C}} p^i(d, \gamma) = c\}$
    $R_c^{i+1} \leftarrow \{d \in A_c^i : p^i(d, c) > \rho\}$
    $\mathbf{w}_c^{i+1} \leftarrow \frac{1}{|R_c^{i+1}|} \cdot \sum_{d \in R_c^{i+1}} \mathbf{w}_d$
  end for
  $i \leftarrow i + 1$
end while
for all $d \in \mathcal{D}_T$ **do**
  $\hat{C}_T(d) \leftarrow \operatorname{argmax}_{c \in \mathcal{C}} \cos(\mathbf{w}_d, \mathbf{w}_c^i)$
end for
return $\hat{C}_T$

Figure 4.1 – *Pseudo-code for the iterative refining algorithm.*

number of non-zero elements. At this extent, we denote with l_D and l_C the mean number of non-zero elements in bags of words for documents and categories, respectively. By definition, we have $l_D \leq |\mathcal{T}|$ and $l_C \leq |\mathcal{T}|$; from experiments is also generally observed $l_D \ll l_C < |\mathcal{T}|$. Cosine similarities for vectors with l_D and l_C non-zero elements respectively can be computed in $O(l_D + l_C)$ time, which can be written as $O(l_C)$ given that $l_D < l_C$.

The construction of the initial representation for categories is done in $O(|\mathcal{D}_S| \cdot l_D)$ time, as all values of all documents representations must be summed up.

In each iteration of the refining phase, the method computes cosine similarity for $N_T = |\mathcal{D}_T| \cdot |\mathcal{C}|$ document-category pairs and normalizes them to obtain distribution probabilities in $O(N_T \cdot l_C)$ time; then, to build new bags of words for categories, up to $|\mathcal{D}_T|$ document bags must be summed up, which is done in $O(|\mathcal{D}_T| \cdot l_D)$ time. The sum of these two steps, always considering $l_D < l_C$, is $O(|\mathcal{D}_T| \cdot |\mathcal{C}| \cdot l_C)$, which must be multiplied by the final number n_I of iterations.

Summing up, the overall complexity of the method is $O(|\mathcal{D}_S| \cdot l_D + n_I \cdot |\mathcal{D}_T| \cdot |\mathcal{C}| \cdot l_C)$, which can be approximated to the component for the iterative phase $O(n_I \cdot |\mathcal{D}_T| \cdot |\mathcal{C}| \cdot l_C)$, with $l_C \leq |\mathcal{W}|$. The complexity is therefore linear in the number $|\mathcal{D}_T|$ of documents, the number $|\mathcal{C}|$ of top categories (usually very small), the mean number l_C of mean terms per category (having $|\mathcal{T}|$ as an upper bound) and the number n_I of iterations in the final phase, which in the experiments presented below is almost always less than 20. This complexity is comparable to the other methods described.

4.4.4 Results

Here are presented the results of experimental runs of the algorithm on the datasets described in §4.3.1. The splits specifically used are the following.

- For two-classes problems on 20 Newsgroups, the splits used in other works and reported in Tables 4.1, 4.2 and 4.3 are used.
- From SRAA, likely to other works, 4.000 documents for each of the four classes have been picked randomly and used across all runs.
- For Reuters-21578, a publicly available[4] split distribution has been used.

[4]Download URL: `http://www.cse.ust.hk/TL/dataset/Reuters.zip`

Regarding the parameters, the maximum number N_I of iterations is fixed to 50, so that the iterative phase usually converges before this limit is reached. Selection of features is always performed by the aforementioned document frequency thresholding with three minimum documents, which is the scheme followed by most other works. The remaining "free" parameters are the term weighting scheme and the similarity threshold ρ for selecting relevant documents in the iterative phase.

To find optimal values for these parameters, we performed tests on *validation datasets*, different from the "final" ones used for comparison with other works. Specifically, for each of the final datasets extracted from the 20 Newsgroups collection, a source and a target domain have been obtained by picking, among the categories normally constituting the source domain, two sub-categories for each top category, one for the source and one for the target.

Through experiments with different local weighting measures, either considered as are or combined with *idf* (inverse document frequency), the following ones generally resulted to give better results:

- *bin.idf* (binary weighting $\cdot$ *idf*),
- *logtf.idf* (logarithmic term frequency $\cdot$ *idf*),
- *ntf.idf* (normalized term frequency $\cdot$ *idf*),
- *cntf.idf* (cosine-normalized term frequency $\cdot$ *idf*).

Plots in Figures 4.2, 4.3 and 4.4 report the accuracy levels obtained in the tests on validation datasets, using the weighting schemes listed above and for different values of the ρ parameter. In all cases, the accuracy maintains optimal values for values of ρ not too far from the minimum possible (which is $1/|\mathcal{C}|$, for example 0.5 with two top categories), while important and usually sharp drops are observed as ρ is overly increased. The range of values of ρ where the accuracy is optimal generally depends from the dataset and the weighting scheme: we observe that *ntf.idf* and *cntf.idf* schemes usually yield a larger range compared to the other two. Selecting the minimum value of ρ generally guarantees very high accuracy, but slightly below the optimal value reachable by tuning the parameter at its best setting.

From the results obtained here, we have to pick default parameter values to run the method on final test datasets. Regarding term weighting,

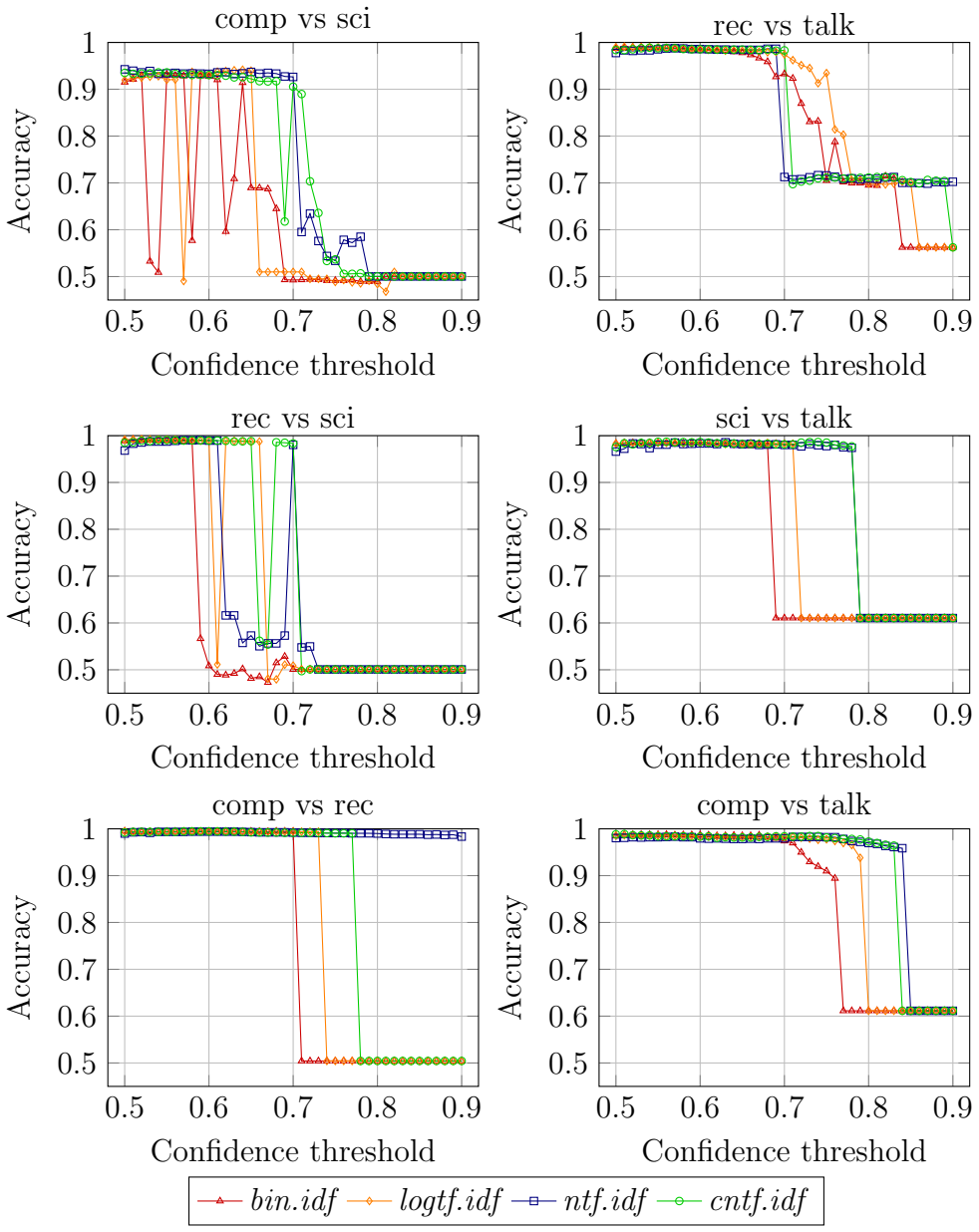

Figure 4.2 – *Accuracy on 20 Newsgroups two-categories validation splits with different term weighting schemes as the confidence threshold ρ varies*

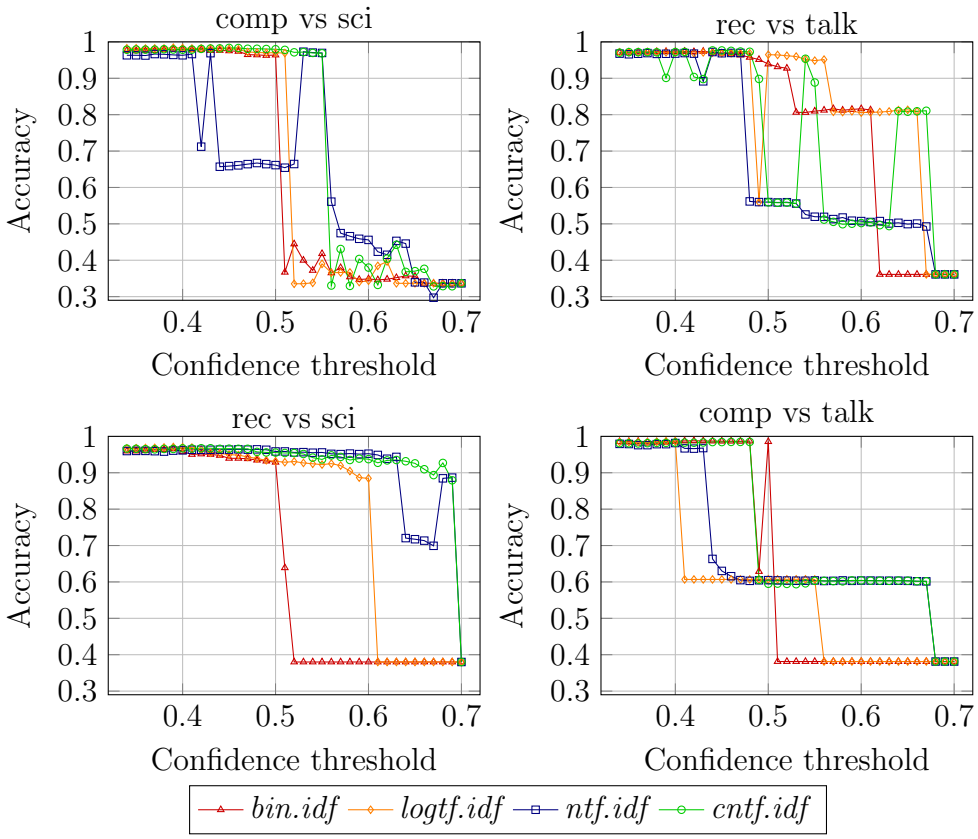

Figure 4.3 – *Accuracy on 20 Newsgroups three-categories validation splits with different term weighting schemes as the confidence threshold ρ varies*

cntf.idf (cosine normalized tf · idf) is chosen for being the scheme yielding the highest overall accuracy, considering an average on all tests, with *ntf.idf* being only slightly inferior. On the other hand, the optimal value for the confidence threshold ρ depends largely from the number $|\mathcal{C}|$ of top categories: we then differentiate the value on the basis of this variable. Always by considering all the tests on the validation datasets, the values of ρ yielding the highest average accuracy on datasets with 2, 3 and 4 top categories are 0.55, 0.4 and 0.36, respectively: we pick those as the default values of the threshold.

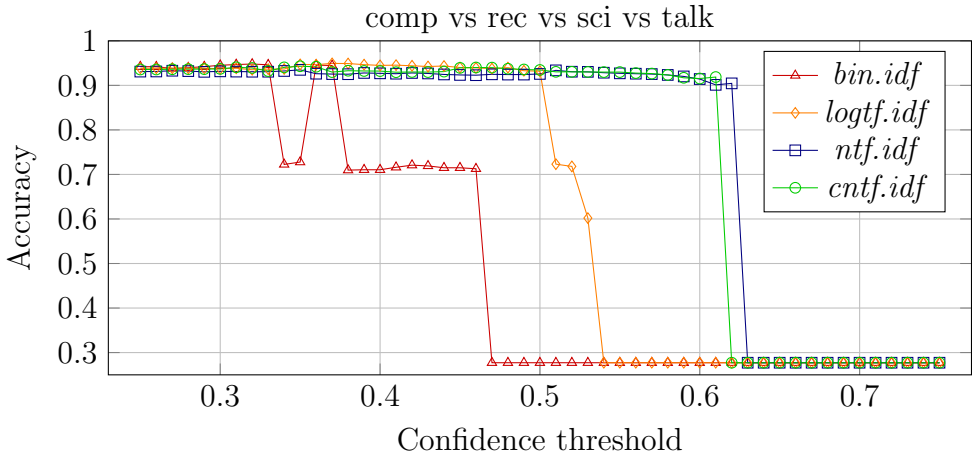

Figure 4.4 – *Accuracy on the 20 Newsgroups four-categories validation split with different term weighting schemes as the confidence threshold ρ varies*

Using these default parameter values, results on effective test datasets are now shown, along with some results for different values.

To evaluate the "difficulty" in classifying documents of each dataset independently from the cross-domain setting, two types of preliminary *baseline* results are given in Table 4.4. These results denote for each dataset a range within which the actual cross-domain accuracy should lie.

- The *no transfer* baseline is the accuracy obtained by classifying target documents directly using the initial categories representations extracted from the source domain. It constitutes a sort of lower bound for the cross-domain accuracy.

- The *same domain* baseline is the accuracy obtained by classifying target documents using categories representations extracted from the target domain itself, corresponding in practice to standard nearest centroid classification using the target domain as both training and test set. It constitutes an upper bound for the cross-domain accuracy.

In the upper bound results, the comparison of performances of the different weighting schemes does not reflect important differences between

Table 4.4 – *Baseline results for classification in cross domain datasets with different term weighting schemes (names shortened for space constraints, so e.g. "bin" stands for "bin.idf").*

	No transfer				**Same domain**			
Dataset	***bin***	***logtf***	***ntf***	***cntf***	***bin***	***logtf***	***ntf***	***cntf***
20 Newsgroups								
comp vs sci	0.828	0.819	0.760	0.786	0.982	0.983	0.988	0.987
rec vs talk	0.653	0.645	0.641	0.647	0.993	0.994	0.998	0.996
rec vs sci	0.843	0.847	0.821	0.829	0.989	0.987	0.990	0.989
sci vs talk	0.763	0.762	0.796	0.791	0.985	0.983	0.989	0.989
comp vs rec	0.874	0.881	0.904	0.902	0.985	0.988	0.992	0.990
comp vs talk	0.967	0.968	0.965	0.962	0.995	0.993	0.995	0.995
comp - rec - sci	0.668	0.674	0.682	0.683	0.960	0.964	0.973	0.972
rec - sci - talk	0.527	0.510	0.487	0.493	0.986	0.983	0.991	0.989
comp - sci - talk	0.709	0.707	0.721	0.705	0.982	0.982	0.986	0.985
comp - rec - talk	0.918	0.923	0.916	0.919	0.982	0.985	0.990	0.988
comp rec sci talk	0.641	0.632	0.587	0.602	0.978	0.977	0.983	0.980
SRAA								
real vs simulated	0.700	0.708	0.723	0.719	0.961	0.961	0.965	0.961
auto vs aviation	0.807	0.822	0.816	0.818	0.972	0.971	0.978	0.976
Reuters-21578								
orgs vs places	0.707	0.722	0.733	0.726	0.915	0.909	0.908	0.905
orgs vs people	0.783	0.768	0.781	0.774	0.928	0.930	0.916	0.932
people vs places	0.632	0.628	0.627	0.635	0.929	0.930	0.926	0.927

them, although *ntf.idf* usually performs better, closely followed by *cntf.idf*. For the lower bound, instead, the variance between results with different schemes is much higher and there is no clearly best scheme: this shows the necessity of cross-domain methods, as normally classifying target documents using a model trained from the different source domain yields poor results.

Table 4.5 reports effective cross-domain results for all datasets comparing the same weighting measures and with default values of the confidence threshold, reporting both the final accuracy and the number of iterations run before convergence. As for the validation datasets, the *cntf.idf* scheme usually appears to perform best: with the other schemes, in at least one

Table 4.5 – *Accuracy and number of iterations for cross-domain categorization on different datasets with different term weighting schemes.*

	bin.idf		***logtf.idf***		***ntf.idf***		***cntf.idf***	
Dataset	**Acc.**	**Itrs.**	**Acc.**	**Itrs.**	**Acc.**	**Itrs.**	**Acc.**	**Itrs.**
20 Newsgroups – 2 classes ($\rho = 0.55$)								
comp vs sci	0.975	7	0.980	8	0.977	13	0.978	9
rec vs talk	0.992	7	0.992	6	0.994	8	0.993	6
rec vs sci	0.982	8	0.980	7	0.984	8	0.985	8
sci vs talk	0.972	15	0.972	9	0.976	9	0.976	9
comp vs rec	0.982	10	0.983	8	0.980	10	0.981	9
comp vs talk	0.990	5	0.990	5	0.991	6	0.992	5
20 Newsgroups – 3 classes ($\rho = 0.4$)								
comp - rec - sci	0.657	16	0.791	24	0.939	16	0.938	29
rec - sci - talk	0.980	10	0.979	11	0.853	12	0.979	12
comp - sci - talk	0.819	10	0.818	12	0.953	15	0.966	13
comp - rec - talk	0.976	11	0.978	9	0.977	7	0.981	8
20 Newsgroups – 4 classes ($\rho = 0.36$)								
comp rec sci talk	0.539	31	0.539	24	0.569	18	0.860	27
SRAA – 2 classes ($\rho = 0.55$)								
real vs simulated	0.950	20	0.949	23	0.932	12	0.936	11
auto vs aviation	0.959	19	0.960	11	0.956	27	0.958	19
Reuters-21578 – 2 classes ($\rho = 0.55$)								
orgs vs places	0.731	9	0.729	6	0.721	13	0.722	9
orgs vs people	0.787	16	0.777	14	0.803	22	0.770	40
people vs places	0.625	19	0.501	39	0.730	44	0.732	25

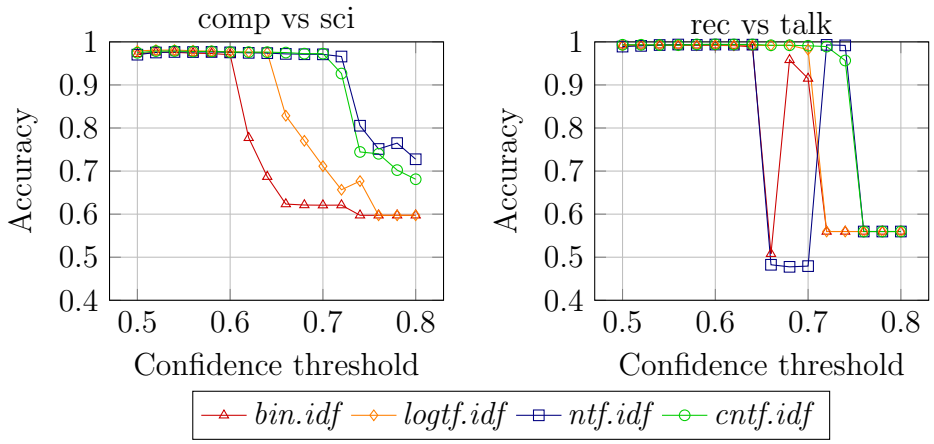

Figure 4.5 – *Accuracy on two 20 Newsgrops two-categories splits with different term weighting schemes as the confidence threshold ρ varies*

case each, important accuracy drops are experienced; among these, *ntf.idf* is the one least subject to this. Regarding the number of iterations, no scheme shows a consistent advantage over the others, although *cntf.idf* often involves a nearly-optimal iteration count other than yielding a good accuracy.

Regarding instead the similarity threshold ρ, plots in Figures 4.5 and 4.6 show how the accuracy varies on some datasets and for different weighting schemes as its value changes, likely to plots shown for the validation datasets. In all cases, the accuracy maintains optimal values for some values of ρ near to the minimum possible, with sharp drops as ρ is increased: this generally confirms observations made above while tuning the parameter.

By observing running tests with high values of ρ and poor accuracies, it is usually noted that the iterative phase stops after few iterations, only two in most cases. In these cases, categories profile generated after the first iteration are generally built from few or even no documents at all and they remain the same in the second iteration, causing the algorithm to terminate. What presumably happens is that the threshold is set so high that no or few target documents reach a sufficient similarity degree with the categories: the variant based on logistic regression presented below partly overcomes this phenomenon.

Table 4.6 – *Results of our method (on rightmost columns) on selected test datasets, compared with those reported by other works: the results in bold are the best for each dataset (excluding baselines).*

	Other methods					
Dataset	**CoCC**	**TPLSA**	**CDSC**	**MTrick**	**TCA**	**I.R.**
20 Newsgroups – 2 classes ($\rho = 0.55$)						
comp vs sci	0.870	**0.989**	0.902	-	0.891	0.978
rec vs talk	0.965	0.977	0.908	*0.950*	0.962	**0.993**
rec vs sci	0.945	0.951	0.876	*0.955*	0.879	**0.985**
sci vs talk	0.946	0.962	0.956	*0.937*	0.940	**0.976**
comp vs rec	0.958	0.951	0.958	-	0.940	**0.981**
comp vs talk	0.980	0.977	0.976	-	0.967	**0.992**
20 Newsgroups – 3 classes ($\rho = 0.4$)						
comp - rec - sci	-	-	-	*0.932*	-	**0.938**
rec - sci - talk	-	-	-	*0.936*	-	**0.979**
comp - sci - talk	-	-	-	*0.921*	-	**0.966**
comp - rec - talk	-	-	-	*0.955*	-	**0.981**
SRAA – 2 classes ($\rho = 0.55$)						
real vs simulated	0.880	0.889	0.812	-	-	**0.936**
auto vs aviation	0.932	0.947	0.880	-	-	**0.958**
Reuters-21578 – 2 classes ($\rho = 0.55$)						
orgs vs places	0.680	0.653	0.682	**0.768**	0.730	0.722
orgs vs people	0.764	0.763	0.768	**0.808**	0.792	0.770
people vs places	**0.826**	0.805	0.798	0.690	0.626	0.732

Values for 20 Newsgroups collection reported by "MTrick" (in italic) actually are not computed on single runs, but are averages of multiple runs, each with an equal set of top categories, where a baseline document classifier trained on source domain and tested on target got an accuracy higher than 65%

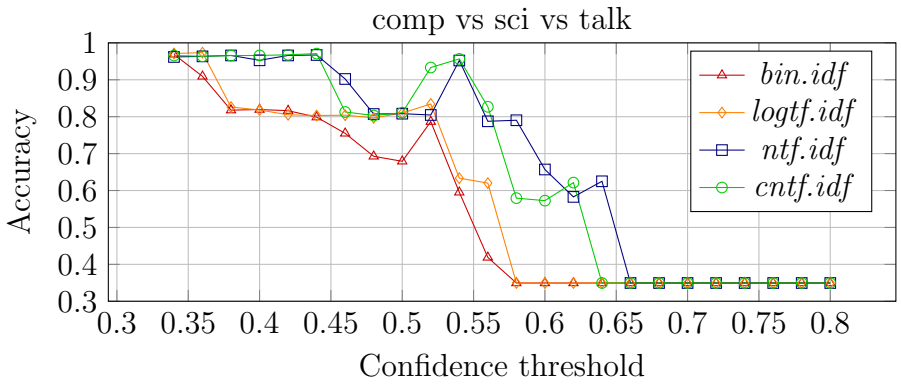

Figure 4.6 – *Accuracy on a 20 Newsgrops three-categories split with different term weighting schemes as the confidence threshold ρ varies*

Table 4.6 compares the accuracy obtained by the iterative refining algorithm using default parameters with those reported by papers presenting other cross-domain text categorization methods, which are namely:

CoCC the co-clustering based approach [24],

TPLSA the topic-bridged PLSA [120],

CDSC the spectral classification method [72],

MTrick the method based on matrix trifactorization [129],

TCA topic correlation analysis [69].

In most tests, the iterative refining approach performs comparably or better than other methods. The most peculiar results are on the Reuters-21578 collection, where the variance between the different methods is much higher[5]: this is probably due to the not fully proper use of ORGS, PEOPLE and PLACES as top categories, as they are independent groups of labels which often co-occur in the documents; also the relatively low results for the upper bounds (Table 4.4) lead to this hypothesis.

[5]Comparing the reported results, it is actually likely that the distribution used in some works has mislabeled datasets, as results given by CoCC, TPLSA and CDSC are very different from the others but would be comparable by permuting the datasets.

Regarding the running times, depending from the considered dataset, the pre-processing phase where documents are read, processed and stored in memory in bag-of-words form is performed in about 5 to 10 seconds, the creation of initial category profiles does not take more than one second and the refining phase requires about 0.5 to 2 seconds for each iteration. Overall, many single tests generally run within 15-20 seconds, with the most long-running ones reaching 30 seconds.

While the given results show a satisfactory performance of the algorithm on the benchmark datasets, in the rest of the section two variants are proposed to make the algorithm less sensible to parameter tuning and to improve its running time. As the two variants operate on distinct parts of the algorithm, they can be applied simultaneously.

4.4.5 Variant with logistic regression

In the base method described above, the result of the cosine similarity between two bags of words is considered directly to estimate their absolute (i.e. not conditioned from others) probability of being related. Anyway, this is used as a rough approximation. For example, this would imply that a document and a category which are very related to each other have representative vectors with a mutual cosine similarity very close to 1, but this is unlikely to happen in practice. Similarly, vectors of unrelated documents and categories should have a cosine similarity close to 0; but often there are some words which are present in almost all documents, granting a minimum degree of similarity to all documents and categories, although lower with respect to related ones.

As an example, the histograms in Figure 4.7 show how related and unrelated document-category couples of the source domain for the *rec vs talk* dataset used in the experiments are distributed across various ranges of cosine similarity. While the average value for unrelated couples is near to 0, that for related couples is very far from the ideal value of 1. Couples extracted from other datasets generally follow similar distributions.

A possible solution is to apply some sort of correction to the "raw" cosine similarity value, so that typical similarity values for couples of related document and categories are mapped to sensibly high probabilities and contrarily typical similarities for unrelated couples are mapped to consistently low probabilities. This can be formalized as applying a suitable function $\pi : [0, 1] \to [0, 1]$ to the cosine similarity between vectors of a document

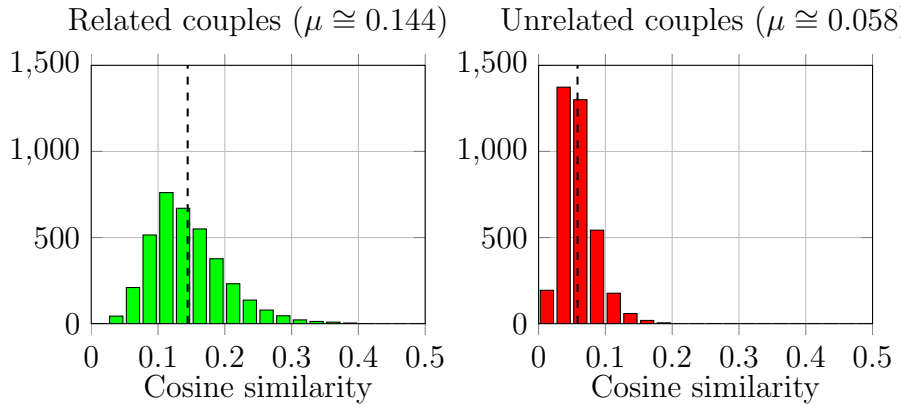

Figure 4.7 – *Distribution of cosine similarity for couples of source documents and categories in the 20 Newsgrops* rec vs talk *dataset, dashed lines indicate averages*

d and a category c to find their final relatedness probability $s^i(d, c)$ (with reference to a generic iteration i of the process).

$$s^i(d, c) = \pi(\cos(\mathbf{w}_d, \mathbf{w}_c^i))$$

To pick a function which adheres to the aforementioned conditions, typical values for cosine similarity for related and unrelated couples should be known. The trivial chosen option is to sample values from couples of the source domain: as the labeling of the documents is known, it is possible to measure the cosine similarity $\cos(\mathbf{w}_d, \mathbf{w}_c^0)$ for all couples $(d, c) \in \mathcal{D}_S \times \mathcal{C}$, which constitute samples of its distribution for both related (those where $C_S(d) = c$) and unrelated couples (the remaining ones).

To generalize these observations, *univariate logistic regression* is used. In general, logistic regression is a probabilistic classification model which can be used as a machine learning technique. In the specific univariate case, a value $\pi(x) \in [0, 1]$ is returned from a single predictive feature with a value $x \in \mathbb{R}$, according to a function with the following form.

$$\pi(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

The goal of the logistic regression algorithm is to find optimal values for

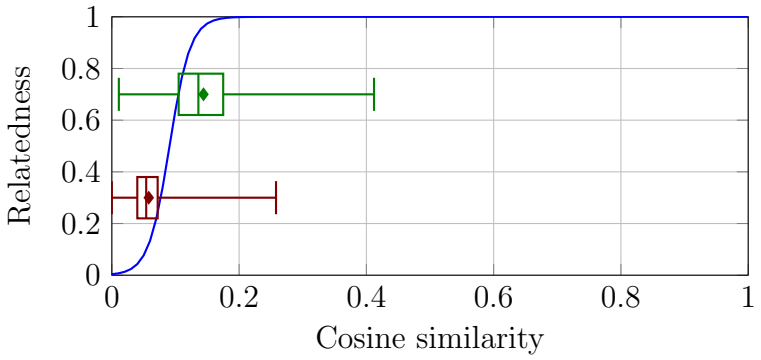

Figure 4.8 – *Logistic function extracted by regression on data of Figure 4.7, for which summary information is reported through box plots (whiskers are total range, box lines are quartiles, diamond is average)*

parameters β_0 and β_1. Given n observations $x_1, x_2, \ldots, x_n \in \mathbb{R}$ respectively labeled with $y_1, y_2, \ldots, y_n \in \{0, 1\}$, the value of the following objective function must be maximized.

$$\prod_{i=1}^{n} \pi(x_i)^{y_i}(1 - \pi(x_i))^{1-y_i}$$

In the problem under analysis, each observation corresponds to a generic source document-category couple $(d, c) \in \mathcal{D}_S \times \mathcal{C}$, the corresponding x value is the cosine similarity of their representations and the y label is 1 if the two are related and 0 otherwise.

$$x_{d,c} = \cos(\mathbf{w}_d, \mathbf{w}_c^0) \qquad y_{d,c} = \begin{cases} 1 & \text{if } C_S(d) = c \\ 0 & \text{if } C_S(d) \neq c \end{cases}$$

As an example, Figure 4.8 shows the function which is obtained through logistic regression on couples of the *rec vs talk* dataset, whose distribution of samples was reported in Figure 4.7. The function returns values above 0.99 for any cosine similarity of at least 0.2, reflecting that such similarity values are reached almost exclusively by related couples.

In practice, the base method is modified as follows. After computing representations for each document and each category from the source domain and before starting the iterative phase, a univariate logistic regression

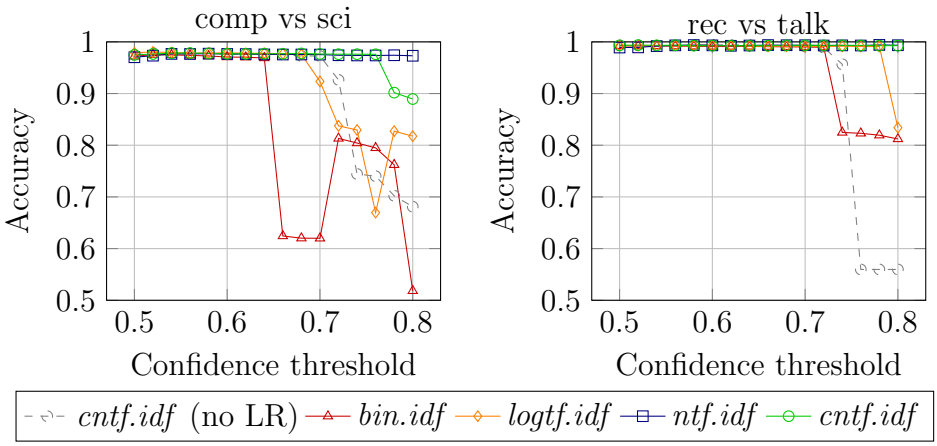

Figure 4.9 – *Accuracy on two 20 Newsgrops two-categories splits with the logistic regression variant and different term weighting schemes as the confidence threshold ρ varies*

model π is trained from observations extracted as specified above from all possible document-category couples $(d, c) \in \mathcal{D}_S \times \mathcal{C}$. Then, during the iterative phase, the model is applied to the result of cosine similarity when computing the absolute relatedness between any document d and any category c.

$$s^i(d, c) = \cos(\mathbf{w}_d, \mathbf{w}_c^i) \qquad \text{is replaced by} \qquad s^i(d, c) = \pi(\cos(\mathbf{w}_d, \mathbf{w}_c^i))$$

For what concerns computational complexity, to fit the logistic regression model, the cosine similarity for $N_S = |\mathcal{D}_S| \cdot |\mathcal{C}|$ pairs must be computed to acquire input data, which requires $O(l_c \cdot N_S)$ time; then the model can be fit with one of various optimization methods which are generally linear in the number N_S of data samples [82]. In practice, in experiments described hereafter, the whole process roughly takes the time of one or two iterations of the refining phase, i.e. not more than 5 seconds.

Experiments on this variant have been carried on using the same parameter values of those on the base algorithm. Plots in Figures 4.9 and 4.10 are the counterparts of those in Figures 4.5 and 4.6 with this variant applied: they show how accuracy varies with the weighting scheme and the

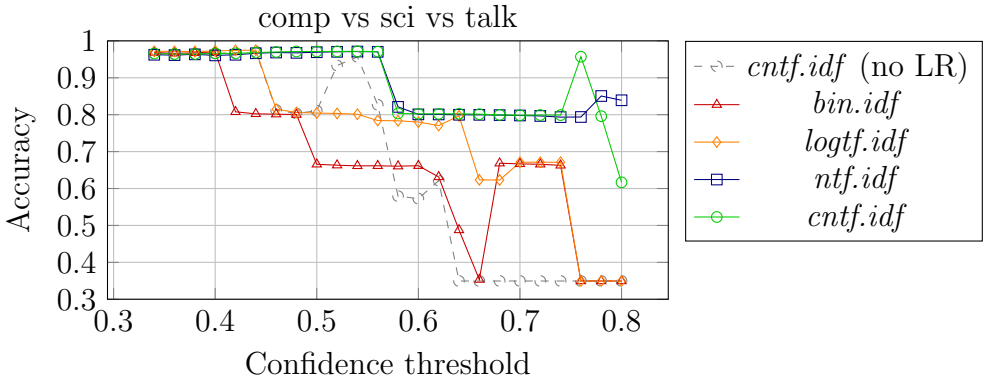

Figure 4.10 – *Accuracy on a 20 Newsgrops three-categories split with the logistic regression variant and different term weighting schemes as the confidence threshold ρ varies*

confidence threshold ρ, in all of them are also shown for comparison results for the base method with the default *cntf.idf* weighting. As can be seen, there is still a drop of accuracy for too high values of the threshold, but this now happens for higher values across all datasets and weighting schemes, as shown by the comparison with *cntf.idf* in the normal case, which is the most "stable" scheme together with *ntf.idf*.

In summary, this variant does not yield significant improvements of the results when parameters are optimally tuned, but guarantees optimal results for a wider range of parameters, derived from a more solid estimation of similarities between document and category representations.

4.4.6 Variant with termination by quasi-similarity

As discussed above, the expected behavior of the iterative refining algorithm is that the representations of categories are progressively improved at each iteration, finally leading to an optimal classification accuracy in the end. However, it is interesting to observe how much the profiles of the categories improve over iterations and specifically how accurate would be classifying document with them.

Figure 4.11 reports a plot of the accuracy levels which would be obtained on different datasets by stopping the iterative phase at a certain

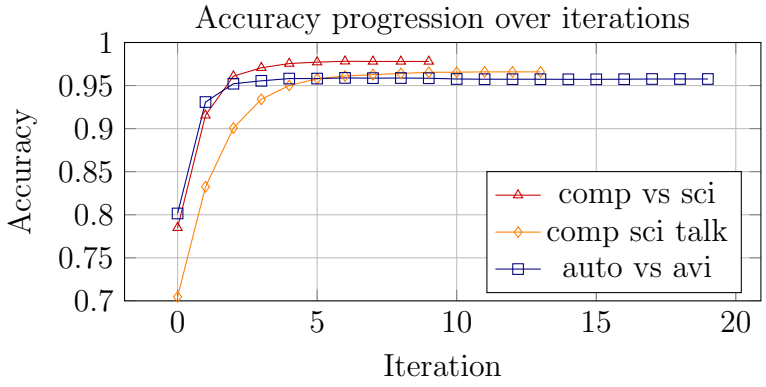

Figure 4.11 – *Progression of accuracy through refining iterations with the* cntf.idf *scheme on different datasets*

number of steps and classifying target documents using the category profiles of that moment; the monitored tests are three of those using *cntf.idf* weighting reported in Table 4.5. Can be noted that the accuracy generally grows rapidly in the first iterations and only has minor fluctuations in later iterations. In the shown cases, 4 iterations grant an accuracy at most 2% below the convergence value, while with 5 iterations the optimal result is within 1%.

In order to further optimize running times while maintaining a good accuracy level, the iterative phase could be stopped before reaching the convergence condition, i.e. before profiles for each category are equal to those of previous step. This could be achieved by setting a lower maximum number N_I of iterations, such as 5 or 10, but the resulting approximation could significantly vary with the dataset and the values of other parameters. For example, referring to Figure 4.11, setting $N_I = 2$ would cause an acceptable approximation of accuracy in the *auto vs avi* dataset (95.2% instead of 95.8%), while for *comp vs sci vs talk* the drop would be more consistent (90.1% instead of 96.6%). An ideal solution should instead grant a similar approximation level across any dataset.

Rather than setting a fixed limit number of iterations, an alternative approach is to set a relaxed convergence condition which is satisfied more "easily" and thus potentially causes earlier termination. An intuitive solution is, for each category $c \in \mathcal{C}$, rather than requiring the new profile $\mathbf{w}_c^{i+1}$

to be *equal* to the previous one $\mathbf{w}_c^i$, to require it to be *close enough*.

The closeness between the two successive representations can be computed trivially by means of cosine similarity. Formally, given a threshold $\beta \in [0, 1]$, the second termination condition of the base method (§4.4.2) is substituted with this:

- representations for all categories have a cosine similarity with their respective previous versions of at least β.

$$\forall c \in \mathcal{C} : \cos(\mathbf{w}_c^{i+1}, \mathbf{w}_c^i) \geq \beta$$

The parameter β can in theory assume any value between 0 and 1; anyway, values to be used in practice are those very close to 1, such as 0.99, 0.999 and similar: smaller values would usually stop the iterative phase too prematurely and cause significant accuracy drops. The value of β in practice sets a trade-off between keeping the accuracy optimal (for greater values) and reducing the number of iterations (for smaller values). It is expected that the effects of varying the parameter β are more consistent across datasets with respect to limiting the absolute maximum number N_I of iterations, so that the new parameter does not need to be tuned for each single dataset.

Again, experiments on this variant are performed with the same parameter values used in the base method. Table 4.7 reports the results, in terms of both accuracy and number of iterations, for the tests with *ntf.idf* and *cntf.idf* already reported in Table 4.5 along with new results obtained setting the β threshold to 0.9999 or 0.999. Results show that the number of iterations is generally reduced, sometimes significantly: in the case with 0.999, it is at least halved down in some cases. For what regards accuracy drop, for $\beta = 0.9999$ it is always within 0.4%, while for β is always under 1%, although with an important exception in *comp vs rec vs sci*. There are even some cases, especially for *Reuters* datasets, where stopping the iterative phase before even results in better accuracies.

Summing up, the goal generally reached with this variant is to shorten the number of iterations of the method and consequently its running time, with a limited drop of accuracy. It can be also noted that, apart from the cited exceptions, the drop of accuracy across different datasets is very consistent for equal values of the β parameter: this allows to set a gener-

Table 4.7 – *Comparison with two weighting schemes of accuracy (A) and number of iterations (I) between the base method and two settings for termination by similarity threshold.*

	ntf.idf						***cntf.idf***					
$\beta \rightarrow$	**base**		**0.9999**		**0.999**		**base**		**0.9999**		**0.999**	
Dataset	**A**	**I**	**A**	**I**	**A**	**I**	**A**	**I**	**A**	**I**	**A**	**I**
20 Newsgroups – 2 classes ($\rho = 0.55$)												
comp vs sci	977	13	977	9	976	7	978	9	978	6	977	5
rec vs talk	994	8	994	6	993	5	993	6	993	5	993	4
rec vs sci	984	8	984	6	984	5	985	8	985	6	985	4
sci vs talk	976	9	976	7	975	5	976	9	976	6	976	5
comp vs rec	980	10	979	5	978	4	981	9	981	5	980	4
comp vs talk	991	6	991	6	990	3	992	5	992	4	992	3
20 Newsgroups – 3 classes ($\rho = 0.4$)												
comp rec sci	939	16	939	13	937	9	938	29	935	21	886	6
rec sci talk	853	12	853	10	852	8	979	12	979	8	978	7
comp sci talk	953	15	952	11	949	8	966	13	965	9	961	6
comp rec talk	977	7	977	5	976	3	981	8	979	4	979	3
20 Newsgroups – 4 classes ($\rho = 0.36$)												
comp rec sci talk	569	18	569	16	573	12	860	27	859	21	859	20
SRAA – 2 classes ($\rho = 0.55$)												
real vs sim	932	12	928	11	922	5	936	11	930	7	928	4
auto vs avi	956	27	960	12	950	4	958	19	962	7	958	4
Reuters-21578 – 2 classes ($\rho = 0.55$)												
orgs places	721	13	721	10	725	6	722	9	722	7	722	5
orgs people	803	22	805	12	820	7	770	40	815	10	824	6
people places	730	44	730	43	773	7	732	25	732	19	756	10

ally predictable trade-off between accuracy and running time independently from the data under analysis.

4.4.7 Discussion

The presented method for cross-domain text categorization based on iterative refining can be seen as an adaptation of nearest centroid classification, where representations of categories of interest are extracted from the source domain and then adapted to the target domain by progressive improvements, so that they get as much close as possible to ideal profiles.

Comparison of obtained results with baselines shows substantial improvement over the results which would be obtained without the iterative phase and, above all, they are rather close to results which would be obtained by knowing in advance the correct centroids for the target domain, indicating that the method is effective in obtaining a good approximation of them. The suggested extensions do not give noticeable improvements in the classification accuracy, but the logistic regression model is useful to make the method more robust with respect to parameters choice, while termination by approximate similarity allows to set a tradeoff between accuracy and running time with a roughly predictable effect. With respect with other cross-domain categorization methods, this obtains better or comparable accuracy, despite of its relative conceptual simplicity (leading to a trivial implementation) and limited running times.

Other than experiments on cross-domain text classification by topic, some tests on datasets used for sentiment analysis were performed, which did not result in satisfactory accuracy levels with respect to the state of the art. Anyway, no specific variants suitable to sentiment analysis have been tested by now: these possibly include the use of n-grams (§2.3.3) and of particular term weighting schemes [86].

Another important improvement on the iterative refining method could be obtained by substituting the centroids and the simple measurement of their distance from documents with a potentially more accurate model. For example, documents selected as representative for a category at an iteration could be used to train a SVM or another machine learning-based model, used then to estimate a relatedness score for all documents. While this would likely increase the computational effort for each iteration, it is possible that this would be compensated from a minor number of such iterations.

The logistic regression model presented in the related variant, telling the degree of relatedness of documents and categories from the cosine similarity of their representations, is trained on documents of the source domain of each experiment and applied to corresponding target documents: a key assumption made in this procedure is that the ideal function correlating measurable similarity and semantic relatedness remains fully or enough valid across the two domains. The work presented in the next chapter starts from a similar hypothesis, applied to a more complex model and to more general text categorization tasks.

Chapter 5

A Domain-Independent Model for Semantic Relatedness

Up to now, known approaches for standard text categorization based on machine learning and for the cross-domain variant have been analyzed. These are generally based on extracting a knowledge model from a training set of pre-labeled documents and using it to classify other documents in the same domain or, in the case of cross-domain methods, a slightly different one.

In this chapter will be introduced an approach differing from the previous ones, based on the creation and utilization of a knowledge model which further generalizes information extracted from the training set to make it potentially applicable to completely different domains. The goal is to investigate the possibility to create a generally valid model for evaluation of semantic relatedness, using text categorization as a concrete application for testing it.

5.1 General idea

Most text categorization methods use machine learning algorithms to infer classification models which allow to predict the membership of subsequent documents to categories seen in the training set. Specifically, the extracted knowledge models embed information which is representative of the specific categories they were trained on and are aimed specifically at recognizing

those categories. Obviously, if new categories to be recognized are subsequently introduced into the system, the training of new knowledge models is required, which may require consistent amounts of time. Even updating a classifier for some category with new documents requires an additional, possibly costly training phase, unless incremental learning methods are used.

In other words, most text categorization methods work by extracting knowledge which is focused on the taxonomy of categories (either flat or hierarchical) known from the training set and can not be easily adapted to different topics which might be introduced at a later time in the same taxonomy or in a different context.

The nearest centroid classifier described in Section 3.6 can be considered less prone to this problem: rather than being based on arbitrarily complex classification models, each category is simply represented by the centroid of the respective relevant training documents. This approach is intuitively more simple in computational terms, as computing a mean of vectors is usually much simpler than building a knowledge model like a decision tree or a SVM. In this case, introducing new categories in the classification context would still require to provide a set of representative documents, but their processing would be very quick.

Anyway, the triviality of the approach, consisting in its basic form in measuring similarity between vectors of known categories and new documents, cannot guarantee the optimal results obtained with more complex machine learning methods: categories must be linearly separable in the space and the number of features must be adequate. As discussed previously, because of these limits, centroid-based methods did not receive focus in recent research on text categorization. The approach proposed here gives in practice a finer method to compare similarly structured representations of documents and categories, which could potentially improve the accuracy of the centroid-based approach while maintaining its flexibility advantages cited above.

Making a step backwards from the vector space model typically used, where documents are reduced to vectors based on a global feature set extracted at training time, each document can be seen as a set of arbitrary words, with associated counts of occurrences and possible derived weights. Likely, categories can be represented in the same way by summing up representations of single documents related to them. Each of the words carries a meaning, which might be related to other meanings expressed by other words in the same document or in other documents. These meanings can be

inferred with the support of semantic knowledge bases, discussed in Section 2.6.

While works based on vector space model leveraging semantic knowledge generally use it to augment the feature space or to modify the weights of terms, here it is used to determine which words are semantically related to each other and also how they are related. When juxtaposing the words contained in a document to those representing a whole category, different types of semantic relationships between them can be identified, which could be indicative of their relatedness as wholes. The identification of such semantic relationships is possible by using a suitable "normative" knowledge base which explicitly indicates these relationships, such as WordNet (§2.6.1).

We expect that, from the semantic relationships between words representing an object (either a document or a category) and those representing another one, the semantic relationship between the two objects themselves can be inferred. Concretely, this implies that the degree of relatedness between a document and a category may be measured from the relationships between their words: roughly, their similarity should be proportional to the quantity of detected relationships. Equal or similar assumptions are made in other works where semantic knowledge is employed. However, in addition to the intensity of the relationship between two objects, we consider that even its type might be distinguished among more possibilities: for example, within a hierarchical taxonomy of categories, one of them may be specifically identified as an ancestor of the other.

The identification of the relationships between objects requires to correctly interpret those between their words: for example, it must be known how many couples of related words must be present in order to consider the objects themselves significantly related. The proposed approach is based on using standard machine learning techniques to capture this general knowledge, rather than training models strictly on documents and categories under analysis.

Consider a set of documents organized in categories: from it representations made of relevant words for both single documents and whole categories can be extracted likely to what was done in methods presented so far. Consequently, arbitrary couples each made of a document and a category can be juxtaposed to check the semantic relationships between their representative terms. Defined a suitable set of "semantic" features, these can be extracted from the information on these relationships to obtain a vector of values. The vector extracted from each couple can then be labeled with

Training set for typical TC methods

example	*features (terms)*				*label*
document	**shot**	**movie**	**song**	...	**category**
doc1	2	1	1	...	MOVIES
doc2	0	1	3	...	MUSIC
doc3	1	2	4	...	MUSIC
...	...	...	...	...	...

terms used to predict **categories**

Training set of couples in our approach

example	*features (term relationships)*			*label*
couple	**synonyms**	**hypernyms**	...	**relation**
doc1-MOVIES	4	2	...	related
doc1-MUSIC	1	3	...	unrelated
doc2-MOVIES	0	2	...	unrelated
...	...	...	...	...

term relationships used to predict
document-category relationships

Figure 5.1 – *Comparison between explanatory examples of training sets for classic text classification methods and for the proposed approach: for simplicity, absolute numbers of occurrences of terms in documents and of relationships in document-category couples are used as features, respectively.*

the known relationship between the document and the category involved: in the most basic case, the document may be either related to the category (i.e. labeled with it) or not.

An adequate number of these labeled vectors can be gathered to constitute a training set, to be given in input to a standard supervised machine learning algorithm, like those seen so far. This yields a classification model where the predictive features are based on semantic relationships between words and the known classes are the possible relationships between documents and categories. Both the final training set and the resulting model references neither specific words nor specific categories used to build the training set: this is shown in Table 5.1, comparing the training set of a typical document classifier to that for the model discussed here.

Assuming the existence of general latent rules binding relationships be-

tween words to those between whole objects, a machine learning algorithm, given a proper training set, can potentially capture these rules in a knowledge model, which could then be applied in any context.

In practice, once such a knowledge model is extracted from a domain, it could be applied to compare representations of documents and categories within it as well as in other domains. Using representations of categories, an arbitrary document can be compared to categories in order to find the ones deemed most related to it, thus classifying the document. For a new domain this requires the creation of representations of categories, which are simply computed as the mean point of relevant documents, but the training of the general semantic knowledge should not need to be repeated.

The construction of the described knowledge model allows to perform classification of documents by comparing them to profiles of categories, likely to the nearest centroid method, but with potentially more precision due to the use of semantic information and with the possibility to identify multiple modalities of relatedness: we leverage this aspect to distinguish documents discussing a general topic from those dealing with a specific branch of it. Additionally, we expect that the model has the property of being equally valid in any domain, so new categories can be introduced at a later time in the same domain or even different domains can be targeted, just by providing representations of relevant categories.

In the rest of the chapter, it is described in detail how to apply this model to text categorization. Over a standard approach, this method offers the advantages described hereafter.

5.1.1 Possible applications

The described approach has some interesting potential applications in practical text categorization tasks.

Suppose to create a domain-independent knowledge model as described above from an adequate training set of labeled documents. A suitable classification system based on the model, using representations of categories already employed for its training, can be immediately set up to classify new documents in the same domain by comparing them with all categories (using the respective representations) to find the most likely ones. To possibly save time, the model may even be trained considering only a part of the categories, as long as they are in a number sufficient for the model to be general enough.

Moreover, it is generally possible to alter the organization of categories while the system is already operational. In particular, new categories can be introduced: once their profiles are given, the system can compare new documents to them in addition to profiles of original categories, so to be possibly labeled with them. In practice, given a set of representative documents, a category can be added efficiently just by computing their centroid. Obviously, it is also possible to remove a category from the set of recognized ones in order to ignore it or to update one's representation to reflect changes in the context.

In the end, as suggested above, the same model as is can be transposed to an arbitrarily different domain where to classify documents. Once profiles of categories are built from a set of labeled documents, always by computing respective means as in the nearest centroids method, a classifier for the new domain based on the already available model can be made operational. Also in this context the taxonomy of categories can be dynamic, with efficient addition of new categories or other changes at a later time.

Consider as a practical use case a news site classifying articles incoming from a constant stream into a taxonomy of categories, for the convenience of the users browsing the site. The taxonomy should include both top-level categories like ECONOMY, POLITICS and SPORTS and specific categories therein about relevant topics and events like FINANCIAL CRISIS, PRESIDENTIAL ELECTIONS and OLYMPIC GAMES. As new events emerge over time, sometimes unexpectedly, categories representing them must be added while the classification system is already operational.

The addition of new categories generally requires to both collect a sufficient number of documents representing the new topics and to train a new or updated classifier to update the knowledge of the system. Using this approach, after collecting the relevant documents, the update of the taxonomy requires negligible computational effort.

5.2 Related work

Related work about text categorization in general has already been exposed throughout Chapter 3. Here some additional works having one or more aspects in common with the approach proposed here are presented.

A prominent characteristic of the presented method is the use of an external semantic knowledge base to obtain required information: this aspect

is common to many other methods for text categorization. As discussed in Section 2.6, used knowledge bases can be either structured and built ad-hoc or not fully structured, thus requiring adequate processing to be used. Among the two, the method presented here requires a knowledge base of the former kind (or anyway incorporating suitable structured knowledge) in order to retrieve semantic relationships between words.

In [56] is tested the leverage of "internal" semantic information: categorization is performed on the Brown Corpus, having words already sense-tagged, using alternatively words or senses as features; the measured difference of accuracy between the two approaches is marginal.

Many works make use of external semantic knowledge by substituting or enriching the representations of documents with concepts expressed by terms, as anticipated in §2.3.4. For example, in [106] WordNet synsets are used as features, weighting them also by respective hypernyms found in the text. In [7] synsets are used alongside words, testing two domain-specific ontologies in addition to WordNet: hypernyms are searched to augment their weights and also basic forms of word sense disambiguation are tested.

In [94] feautres corresponding to WordNet synsets are added to those for words and their weights are adjusted according to the subsumption hierarchy, by increasing that of each general synset according to those of hyponyms; moreover, also this method like the one proposed here is also based on creating category representations, which however are compared to documents simply by means of cosine similarity.

Alternative approaches also exist to leverage semantic information rather than adding features. As already cited in §3.5.2, in [110] WordNet is used for a semantic kernel to be employed in a standard support vector machine learner. In [79] is instead proposed a term weighting scheme based on semantic knowledge and also on category names.

Other works make use of suitable unstructured data, which is processed to obtain a subsumption hierarchy of concepts and other information similar to that given by WordNet. In [40] is tested the introduction of additional features extracting hundreds of thousands of concepts from DMOZ; the same authors also test an equivalent method using Wikipedia instead [41]. Similarly, in [114] is performed unsupervised classification by extracting categories from generalized world knowledge.

5.3 General working scheme for text categorization

Here the general idea exposed above is made concrete in a high-level procedure to build the domain-independent model and to use it to classify documents. In later sections, specific methods and evaluations thereof for flat and hierarchical text categorization based on this general scheme are given.

5.3.1 Semantic knowledge base

In both training and classification phases, to know the semantic relationships existing between words read in documents, some sort of semantic knowledge base is needed.

In the following, such knowledge base will be considered formally as a function $\Theta : \mathcal{W} \times \mathcal{W} \to [0,1]^{|\mathcal{R}|}$, which maps to an asymmetric pair of words $(w_a, w_b) \in \mathcal{W} \times \mathcal{W}$ (considered here as generic strings of characters) a vector $\Theta(w_a, w_b)$ of values between 0 and 1 giving for each known *relationship class* of a set $\mathcal{R}$ a score indicating the degree with which that type of relationship holds (or may hold) from w_a to w_b. Intuitively, the elements of $\Theta(w_a, w_b)$ indicate with weights what are the possible ways (if any) in which w_a is related to w_b. As seen in the following, Θ will represent specifically an algorithm which searches for semantic relationships in the concrete knowledge base, which will be also referred to as *semantic search algorithm*.

Considering as output a vector of weights rather than a single relationship class or a subset thereof or a single relatedness weight, allows for more expressiveness of the possible relationships between terms.

- It is considered that more than one relationship class could be considered at once, because two words might be linked by more than one type of relationship, possibly because of the multiple meanings they might have.
- Giving a vector instead of a set allows to express *fuzzy* relationships, which can be useful for different reasons, such as expressing uncertainty about a relationship class (for example if a word has more possible senses) or indicating a "weak" relationship.

The following are a couple of examples of the points above.

- Intuitively, wheel is a meronym of bicycle as wheels are *part of* a bicycle. However, wheel is seldom used with the very same meaning of bicycle (i.e. the whole vehicle), so they may be also regarded as synonyms. Using fuzzy relationships, a small (but non-zero) value could be assigned as a weight for synonymy, in addition to the high value for meronymy.
- The word dog is an hyponym of both mammal and organism (among many others), but the hypernymy/hyponymy relationship with the latter can be considered to be weak, because of the high generality of the term.

Supposing to have an algorithm which instead returns a single relationship class from a finite set or a subset thereof, it can be trivially adapted to this scheme by considering a function which returns for each input pair of words a vector whose values are 1 for classes of found relationships and 0 for other classes.

While specific algorithms exist to compute a single score denoting the degree of relatedness between words (see §5.4.2), the use of multiple scores allows to distinguish different ways in which words can be related: this can be particularly useful for example to distinguish similarity (as between dog and cat) from hypernymy and hyponymy (as between dog and animal).

The semantic function Θ is used in the specific algorithm which compares representations of documents and categories, presented in §5.3.3. As a concrete knowledge base, WordNet (§2.6.1) is here considered, although it can't be used directly as a function of the required type: this issue will be discussed thoroughly in Section 5.4.

5.3.2 Model training

As an input, the training of the domain-independent model requires a set $\mathcal{D}_T$ of documents labeled with categories of a set $\mathcal{C}_T$. In this general schema, it is not considered explicitly how documents are assigned to categories and whether these are organized in a hierarchy. Instead, all this information is abstracted with a general function $\Delta_T : \mathcal{D}_T \times \mathcal{C}_T \to \mathcal{K}$, which maps to a generic couple of a document d and a category c a label indicating how they are related.

As a first step, a representation of relevant words for each document $d \in \mathcal{D}_T$ must be obtained. Following the typical bag-of-words approach,

distinct words are extracted from text and weighted proportionally to their recurrence in single documents and in whole collection, according to any technique described in §2.4.3.

However, within this method, differently from typical text categorization approaches, no global set of features is selected: each bag of words is considered as an isolate entity rather then being a vector with weights for a defined set of words. Formally, the representation of each document d is constituted by a function $\omega_d : \mathcal{W} \to \mathbb{R}^+$ mapping each word contained in the document to its corresponding weight and any other possible word to 0. This formalization allows to effectively decouple all the parts of the method from the specific dictionary of words involved.

Other than from documents, also from each category $c \in \mathcal{C}_T$ is extracted a representation ω_c of the same form. The most obvious approach, already seen for nearest centroid classification, is to compute a mean of the representations of documents related to the category itself. In hierarchical classification, as discussed later in its own section, these means should also include documents of some sub-categories.

At this point, to create a training set for the model, labeled vectors must be extracted from couples of documents and categories. The first step is to pick a set of such couples from the available ones in the training corpus. According to some specific method, which depends from the type of categorization to perform, a set $E \subseteq \mathcal{D}_T \times \mathcal{C}_T$ of *example couples* is selected. Generally, this set should be representative of the different possible document-category relationships defined in $\mathcal{K}$, with enough instances for each of them.

For each example couple $(d, c) \in E$, a vector $\mathbf{p}_{d,c} \in \mathbb{R}^n$ must be obtained and then labeled with the known relationship $\Delta_T(d, c) \in \mathcal{K}$ between the document d and the category c. The extraction of a vector from each couple is a task performed by a so-called *semantic matching algorithm* (SMA), which will be described briefly.

In the end, the training set of labeled vectors is passed as input to a standard machine learning algorithm, which infers the needed knowledge model, which classifies subsequent vectors obtained by the SMA into classes in $\mathcal{K}$. In order to quantify the likelihood of relatedness between documents and categories, the learning algorithm must output a *probabilistic* classifier, which rather than indicating a single class from the set $\mathcal{K}$ in response to an input vector, returns a distribution of probabilities between the different classes.

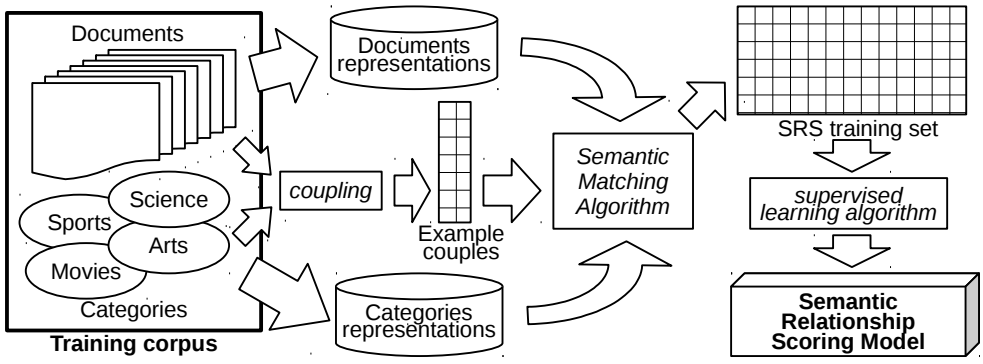

Figure 5.2 – *Schema of the process to train the semantic relationship scoring model*

This classification model, denoted with $\Omega : \mathbb{R}^n \rightarrow [0,1]^{|\mathcal{K}|}$ and referred to as *semantic relationship scoring model* (SRSM), is subsequently used in the document classification phase to compare incoming documents with target categories, as explained later. The whole described process to obtain the SRSM is summarized in Figure 5.2.

5.3.3 Semantic matching algorithm

The purpose of the semantic matching algorithm, denoted with $\Psi : (\mathcal{W} \rightarrow \mathbb{R}_0^+) \times (\mathcal{W} \rightarrow \mathbb{R}_0^+) \rightarrow \mathbb{R}^n$, is to extract a vector $\mathbf{p}_{d,c} \in \mathbb{R}^n$ from the analysis of the semantic relationships between words characterizing a document d and a category c, which are taken from their respective structured representations ω_d and ω_c.

The working of the SMA is influenced by two positive integer parameters n_{WpD} and n_{WpC}, which indicate the number of words to be considered for representations of each document and of each category, respectively. The limit set by these parameters is posed principally to reduce the number of queries to the Θ function for semantic relationships. Obviously, for documents and categories represented by less than n_{WpD} or n_{WpC} words, all of them are considered. The numbers of words considered by the SMA for d and c are denoted in the following with l_d and l_c respectively.

$$l_d = \min(n_{WpD}, |w \in \mathcal{W} : \omega_d(w) > 0|)$$

$$l_c = \min(n_{WpC}, |w \in \mathcal{W} : \omega_c(w) > 0|)$$

As only a part of the words (if more than n_{WpD} or n_{WpC}) representing the document and the category are considered, it is desirable to pick those having more importance within them. For this, words of each bag are put in order by descending weights[1] and throughout the SMA are considered only the first l_d for the document and the first l_c for the category, denoted respectively with $t_1^d, t_2^d, \ldots, t_{l_d}^d$ and $t_1^c, t_2^c, \ldots, t_{l_c}^c$.

$$\omega_x(t_1^x) \geq \omega_x(t_2^x) \geq \ldots \geq \omega_x(t_{l_x}^x) \geq \omega_x(\tau)\, \forall \tau \in \mathcal{W} - \{t_1^x, t_2^x, \ldots, t_{l_x}^x\}$$

At this point, every possible couple (t_i^d, t_j^c) of one of the considered words for d and one for c could be compared. Anyway, to further reduce the number of comparisons with a limited loss of information, a subset $T_{d,c}$ of most relevant couples can be considered. Following some experiments, a criterion which resulted to grant a reduction of comparisons with a negligible loss of final classification accuracy is the following.

$$T_{d,c} = \left\{ (t_i^d, t_j^c) : \left(\frac{i-1}{l_d}\right)^2 + \left(\frac{j-1}{l_c}\right)^2 < 1 \right\}$$

In practice, picturing the possible couples disposed in a grid, this corresponds to discarding those which lay outside of an ellipse having its center at (0,0) and passing for $(0, l_d)$ and $(l_c, 0)$, which are about $\pi/4$ of the total. In this way, the least relevant terms of each document (within the limit set by n_{WpD}) are still coupled to those most relevant for each category and vice versa, but pairs where both words are not much important are discarded. This is shown graphically in Figure 5.3.

For each selected couple $(w_d, w_c) \in T_{d,c}$, the vector $\Theta(w_d, w_c)$ weighting the semantic relationships between the two is computed. The vectors for all the considered couples are then weighted by the product of the weights of the two involved words before being summed up. The sum is then normalized by dividing it for the length of the vectors constituted by weights of considered words of both document and category.

[1] No specific indication is given on how to break ties between words with equal weights, as with most commonly used weighting schemes (especially those based on combination of factors, such as *tf.idf* and alike) they very rarely have any influence on the results.

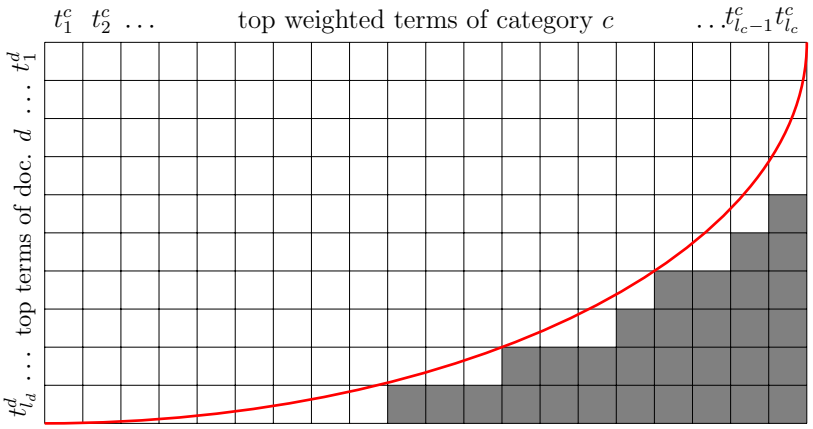

Figure 5.3 – *Schema of selection of relevant pairs of words between representations of a document d and a category c: pairs corresponding to white cells are considered in the computations, while those with a gray cell are discarded.*

$$\mathbf{r}_{d,c} = \frac{1}{\sqrt{\sum_{i=1}^{l_d} \omega_d(t_i^d)} \cdot \sqrt{\sum_{j=1}^{l_c} \omega_c(t_j^c)}} \cdot \sum_{(t_i^d, t_j^c) \in T_{d,c}} \omega_d(t_i^d) \cdot \omega_c(t_j^c) \cdot \Theta(t_i^d, t_j^c)$$

The formula is somehow based on cosine similarity (§2.2.1). Notably, if the number of considered words would not be limited to n_{WpD} and n_{WpC} and the search of semantic relationship would simply result in $\Theta(w_a, w_b) = (1)$ if $w_a = w_b$ and (0) otherwise, the formula would return a vector with a single value, which would correspond to the cosine similarity of the bags of words.

This vector $\mathbf{r}_{d,c}$ constitutes a summary of which semantic relationships occur between the relevant words of d and c and how much they are present across it, also considering how much each word is relevant. The final output of the SMA is constituted by this vector along with the sum of its values as an additional feature, which can be helpful for certain learning algorithms. The total count of features is then $|\mathcal{R}| + 1$.

$$\mathbf{p}_{d,c} = \Psi(\omega_d, \omega_c) = \mathbf{r}_{d,c} \oplus \text{sum}(\mathbf{r}_{d,c})$$

This is the vector returned by the SMA in response to two representations ω_d and ω_c: in the training phase described above it is then labeled with the actual relationship between d and c to be put in the training set for the semantic model, while in the classification phase, as discussed below, is passed to the existing model to predict such relationship.

5.3.4 Classification

Once a semantic relationship scoring model $\Omega : \mathbb{R}^{|\mathcal{R}|+1} \rightarrow [0,1]^{|\mathcal{K}|}$ is trained according to the procedure described above, it can be combined with the semantic matching algorithm $\Psi : (\mathcal{W} \rightarrow \mathbb{R}_0^+) \times (\mathcal{W} \rightarrow \mathbb{R}_0^+) \rightarrow \mathbb{R}^{|\mathcal{R}|+1}$ to obtain a function which, given the representations of any document d and any category c, returns a distribution of probability among the possible relationships in $\mathcal{K}$ between them. In practice, most likely relationship for a (d, c) couple is the having the highest probability in the distribution obtained by $\Omega(\Psi(\omega_d, \omega_c))$.

The combination of SMA and SRSM can be used within a document classifier, which labels input documents with a set $\mathcal{C}$ of *target* categories, which can be either the same used to train the model or a different one.

In the proposed approach, classification is based in general on comparing representations of the input document d with those of all or some of the target categories, according to the specific method used. This requires representations of the target categories, which should be obtained from a set of documents consistently labeled with them: this is denoted with $\mathcal{D}_P$ and referred to as *profiling set*. Another possibility, not tested here, would be to provide manually built representations, made up with some picked keywords.

Once the representations are given, classification of new documents can be performed. The specific procedure to classify a document varies between the flat and the hierarchical case and will be described in detail in the respective sections. In both cases, it is possible to change the taxonomy of categories subsequently, by providing new profiles for categories to be added or updated.

5.3.5 Computational complexity

Throughout both the model training and the classification phase, computation generally consists into comparisons between representations of doc-

uments and categories, which are carried out by searching semantic relationships between the respective relevant words: so, the final complexity of the method largely depends from the complexity of the adopted semantic search algorithm.

In both phases, is required to extract profiles of categories: this process slightly changes between flat and hierarchical classification, but in both cases can be assumed to be linear in the number of documents to be processed and in their average number of words. However, in practice, this part takes negligible time with respect to the rest of the process.

In the training phase, the number of comparisons depends from the number $|E|$ of example couples, which has $|\mathcal{D}_T| \cdot |\mathcal{C}_T|$ as an upper bound, but is in practice limited by other parameters which will be presented in the methods for flat and hierarchical categorization. For each couple, semantic relationships are searched for a number of pairs of words within $n_{WpD} \cdot n_{WpC}$. Overall, the construction of the training set for the SRSM requires a number $O(|E| \cdot n_{WpD} \cdot n_{WpC})$ of queries to the semantic search algorithm. Regarding the training itself of the model, the complexity depends from the specific learning algorithm used, which is usually linear or superlinear with respect to the number of training instances, which in this case is $|E|$.

The complexity of the classification phase changes between flat and hierarchical classification, but in both cases requires, for each document to be classified, its comparison with profiles of either all possible categories of a set $\mathcal{C}$ or a subset thereof: likely to above, this requires a number $O(|\mathcal{C}| \cdot n_{WpD} \cdot n_{WpC})$ of searches for semantic relationships.

5.3.6 General experiment setup

In both the flat and the hierarchical case, accuracy of the method has been evaluated experimentally on some benchmark datasets. Here are described details about experiments which are common between the two cases; more detail are given in the related part of the respective sections.

In both cases, each text corpus used as a benchmark is priorly split in a training set and a test set. Generally, in experiments for other text categorization methods, the training set is used as the input to a machine learning algorithm which extracts one or more classification models, which are then applied to the test set to compare their response with the known actual categories for each test document.

Table 5.1 – *Usage of training and test splits of each dataset*

Phase	Training split	Test split
(1) SRSM training	training set ($\mathcal{D}_T$) Extraction of training set from example couples	not used
(2) Documents classification	profiling set ($\mathcal{D}_P$) Creation of category profiles	test set ($\mathcal{D}_E$) Documents to be classified

Within the method presented here, two decoupled tasks are considered: the training of a semantic model and its use to classify documents. Given the domain-independence assumption, it is implied that the text collections used to perform the two tasks can be different: documents of a dataset can be classified by leveraging the SRSM extracted from another dataset. However, as discussed above, category profiles used when classifying target documents must reflect the exact topics of the documents, so the same dataset of the test documents is used to ensure this.

So, in practice, when training a SRSM throughout the experiments, the training split of a selected dataset is used as the set of documents $\mathcal{D}_T$ from which the training set of couple vectors is built. To test the method, given a built model, the training split of a dataset is used as the profiling set $\mathcal{D}_P$ from which category representations are built, then documents of the test split of the same dataset are used to test the method, comparing categories predicted by the classifier with the actual ones. This organization is summarized in Table 5.1.

To extract bags of words for documents, here in form of functions $\mathcal{W} \to \mathbb{R}_0^+$ rather than of vectors, a pre-precessing step is required. In the following, this is considered to be composed of the following operations for each document (first 4 steps are the same as in §4.4.2):

- words are extracted by splitting on all non-letter characters (spaces, periods, dashes, ...),
- each word is case-folded (all letters are turned lowercase) for uniformity,
- words shorter than 3 letters are removed,

- words appearing in a predefined list of 671 stopwords are removed[2],
- lemmatisation (§2.3.1) is applied to reduce each inflected word to its base form.

The lemmatisation algorithm used is based on WordNet, so that words effectively appearing within it are obtained where possible: this aids in using WordNet itself to search for semantic relationship between them. The use of a stemming algorithm, usually employed in other works, would make more difficult the search for relationships in WordNet, as many stems would not be found in WordNet, not being complete words by themselves.

As semantic relationships between relevant terms of documents must be searched, it would be potentially useful to either use words tagged with information about their meaning or to directly extract concepts instead (§2.3.4). While this could be obtained with techniques for part-of-speech tagging and word sense disambiguation (described in Section 1.4), it is chosen to take instead words as they appear to keep the document processing phase as simple as possible, so that also the computation of category profiles from still unprocessed documents can be run fast.

Terms are weighted by *cntf.idf*, the product of cosine normalized term frequency and inverse document frequency (§2.4.3). As cited above, no feature selection is performed, as documents and categories are considered to be represented as lists of all the words they contain with associated weights rather than as vectors, so it is not necessary to establish a global set of features.

Other parameters of the method will be discussed in detail in the sections related specifically to variants for flat and hierarchical categorization.

5.4 Search of semantic relationships

In the previous section, a semantic knowledge base has been presented as a generic function mapping a couple of arbitrary words to a vector expressing potential semantic relationships holding between them. As a concrete information source for the needed semantic knowledge, WordNet is chosen to be used given its large diffusion in the related literature. However, its usage in the method presented above is not straightforward.

[2]This is the same list cited in §4.4.2, also see footnote there.

5.4.1 Use of WordNet

As discussed in §2.6.1, WordNet is a lexical database for the English language containing a set of lemmas divided between four part-of-speech classes and grouped into synsets of synonym words, with various types of semantic relationships defined between synsets and between words. This database can be used as the knowledge base used in the semantic matching algorithm, but some expedient is necessary.

Firstly, the required knowledge base must define relationships between *words*, intended as generic strings of characters; WordNet defines instead relationships between *words* intended as specific instances of lemmas inside synsets and between synsets themselves. A simple solution would be to "transpose" these relationships to lemmas, which are then considered as the words found in documents. Specifically, setting $\mathcal{R}$ to the set of 28 possible classes of relationship defined in WordNet, two words (lemmas) w_a and w_b can be considered to hold a relationship of type $r \in \mathcal{R}$ if and only if:

- two (WordNet) words, each instance of one of the two lemmas, are related by r, *or*
- two synsets, each containing one of the two lemmas, are related by r.

If one or both of the words are not found in the WordNet database, no relationship can be found between them, so the Θ function returns a null vector.

With these rules, the relationships defined by WordNet between two words can be found. However, this does not include the synonymy relationship, which is not represented explicitly in Wordnet but implicitly exists between two words in the same synset and can be transposed like others to the corresponding lemmas. Other than this, two words extracted from different sets (like one from a document and one from a category profile, as happens in the method) may also be identical. For this, the two symmetric relationship classes *equality* and *synonymy* are introduced in the set $\mathcal{R}$ alongside relationship explicitly defined in WordNet, so that:

- two words are *equal* if they are the same string, even if WordNet does not contain a lemma matching the string) *and*
- two words (lemmas) are *synonym* if are not equal and at least one synset contains them both as words.

With these additions, all potential classes of relationships between words found in documents are covered. Anyway, there is still a limit in applying the above rules for relationships between synsets and between words: the missing identification of indirect relationships.

Consider for example the nouns bicycle and vehicle: intuitively, the first should be identified as a *hyponym* (i.e. "a specific type") of the second and conversely the second as a hypernym of the first. Anyway, WordNet does not explicitly consider a relationship between these two words. Instead, bicycle has wheeled vehicle as its hypernym, which is in turn a hyponym of vehicle. An even more representative example is the similar relationship between dog and animal, which is obvious to a person, but WordNet organizes living organisms likely to standard biological classification, so the synsets for dog and animal are separated by the following chain of hypernyms, in order of decreasing specificity: canine, carnivore, placental, mammal, vertebrate, chordate.

In order to save space, the WordNet database only stores primitive relationships, while indirect relationships like the examples above must be inferred from the primitive ones. One possibility would be to precompute an expanded version of WordNet with all the indirect relationships made explicit.

Anyway, with the goal to maintain a limited memory usage, methods to search indirect relationships "on the fly" from the primitive WordNet graph are employed. Should be noted that, given the high number of comparisons between words which are made while training and using the semantic model, it is critical for any algorithm used to search relationships between words to run very fast.

In the rest of the section, possible solutions to realize such an algorithm are discussed, along with how to consider chains of relationships involving different types of pointers. For example, if a word w_a in WordNet is a *meronym* of an *hyponym* of another word w_b, which is the actual general relationship between w_a and w_b, if any?

5.4.2 Existing approaches

Some literature exist where the problem of inferring non-trivial semantic relationships between words starting from a "primitive" knowledge base like WordNet is studied.

Many works (especially earlier) only considered the hypernymy/hyponymy relationships between synsets, being the most common and important of WordNet semantic relationships. In general, most works only consider a part of the type of pointers defined in WordNet, which have very different frequencies.

Existing works regarding the evaluation of relatedness between words using a semantic knowledge base are mostly focused on computing a single score for any couple of words, rather than a combination of scores for all possible semantic relationships. Specifically, proposed methods generally address one of two different concepts [12].

- *Similarity* of two words refers to the degree of overlap in their meaning: this may be given from synonymy (full overlap) or from hypernymy/hyponymy (partial overlap). For example, tiger and cat are quite similar (they are both felines, which is a rather specific concept), while money and bank are dissimilar (one is a medium of exchange, the other is an institution).

- *Relatedness* of two words refers to how much they are related generally by association of ideas; this is a far more generic concept, including virtually any type of semantic relationship. For example, money and bank are not similar (see above), but any person is likely to consider them strongly related, as they share the domain of discourse.

Given a primitive graph of direct relationships between words, a basic intuition is that similar words have a short distance between them in the graph: in [101] this type of approach is tested on the MeSH (Medical Subject Headings) taxonomy.

Among those based on WordNet, the similarity measure proposed in [103] is the maximum *information content* of synsets subsuming (i.e. being hypernyms of) both the two analyzed words: the information content of a concept x is lower as the probability of encountering any concept subsumed by x is higher. In [48] hypernymy, hyponymy and antonymy relationships of WordNet are considered: the relatedness between two words depends both from the number of links between them and from how much time the type of link ("direction") changes throughout the path; relatedness is considered anyway only for some specific patterns of these direction changes. The method proposed in [96] is able to consistently compare arbitrarily grained

units of texts (from single senses to whole documents) by reducing them to probability distributions of senses and measuring similarity between these.

Recalling the distinction given in §2.6, other than using WordNet or similar *ad hoc*, structured knowledge bases, an alternative approach is to extract information from a large corpus (collection) of text, useful to infer relatedness of words by their frequency of co-occurrence or other statistical properties. For example, in *Explicit Semantic Analysis* [41], the meaning of either single words or texts is mapped to a high-dimensional space of concepts extracted from Wikipedia: relatedness can be then simply computed as the (cosine) similarity between corresponding vectors. In [2] a comparison of different methods of both approaches is given. There also are hybrid solutions: in [51] the distance between two terms depends from their most specific common hypernym in WordNet and is computed from information content of the three, statistically estimated from a corpus of text.

The evaluation of these methods can be difficult, as actual similarity and (especially) relatedness between words can be a rather subjective judgment. Some small collections of pairs of words with associated similarity or relatedness scores exist, which have been compiled by averaging judgments given by multiple persons: an example of these is the *WordSimilarity-353 Test Collection*[3] [36]. A method can be evaluated by comparing its results for these pairs with the actual ones: accuracy is usually measured by the Spearman correlation between actual and predicted scores, which indicates accordance between the order given to pairs by the scores rather than between the scores themselves. Another possible approach is to evaluate a measure by observing the performances of a certain task where it is used: by keeping constant other parameters of this task, different measures can be compared.

5.4.3 Hypernyms-based algorithm

Here is proposed a relatively simple algorithm to search relationships between words which partly overcomes the limit of the simple search of direct relationships of not retrieving indirect ones. This algorithm is summarily based on searching for relationships between given words and also between

[3] `http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/wordsim353.html`

respective hypernym synsets up to a certain maximum number of steps, which is a parameter of the algorithm denoted with M_H.

The set $\mathcal{R}$ of possible relationships recognized by this algorithm, as discussed in §5.4.1, is composed by all 28 pointer types defined in WordNet with the addition of *equality* and *synonymy*. This basic algorithm does not weight returned relationships, so the resulting vector only contains zeros and ones according to the found relationships (either none, one or more than one).

Firstly, if the two compared words w_a and w_b are equal, the algorithm terminates returning only the equality relationship. Otherwise, the sets S_a and S_b of all synsets containing w_a and w_b are retrieved: if one or more synsets appears in both sets ($S_a \cap S_b \neq \emptyset$), it means that those synsets contain both w_a and w_b and the algorithm stops by labeling them as synonyms.

At this point, if the algorithm did not terminate, from each S_x (with $x = a, b$) is extracted the *hypernyms set* of synsets $H_x \supseteq S_x$, made up from all the synsets which can be reached within M_H steps starting from synsets of S_x by following hypernymy pointers (including synsets of S_x themselves). At first, these sets are used to check for indirect hypernymy/hyponymy relationship between the two words: specifically, if at least one of the synsets of S_a is contained in H_b, i.e. $S_a \cap H_b \neq \emptyset$, then w_a is labeled as hypernym of w_b; conversely, if $S_b \cap H_a \neq \emptyset$ then w_a is labeled as hyponym of w_b. Regardless of found relationships, the algorithm then proceeds with the next step.

As hypernymy conceptually represents a *is-a* relationship, synsets are considered to inherit the properties of their hypernyms, including relationships with other synsets. For this, direct relationship are searched between H_a and H_b rather than between S_a and S_b how would happen in a simple method. This allows to extend some relationships involving general synsets down to more specific synsets. For example, the part meronymy pointer from wheeled vehicle to wheel is inherited by bicycle, being it a direct hyponym of wheeled vehicle: this algorithm so returns the intuitively held meronymy relationship between bicycle and wheel which would not be found by just looking for direct relationships. After this search, the algorithm terminates by returning a vector of ones for found relationships and of zeros for all the others.

This is a relatively simple algorithm to find some indirect relationships, based on making the two searched terms "inherit" pointers from their hypernyms. As most synsets have a single hypernym each, searching them for

each input word has a roughly linear complexity with respect to the number M_H of considered levels, while search of relationships among all hypernyms takes about quadratic time. The algorithm presented next starts from the same principle of examining hypernyms of input words, using an improved method and extending it to some other relationship types.

5.4.4 Extended search algorithm

Here is presented an improved algorithm for search of indirect relationships in WordNet, where a new set $\mathcal{R}$ of relationship types is introduced and fuzzy relationships are considered. Specifically, 14 types of relationships listed below are considered, so that the output of the algorithm is a vector of 14 values denoting the weight with which each of them holds.

- equality
- synonymy
- hypernymy
- hyponymy
- coordination
- holonymy
- meronymy
- co-occurrence
- similarity
- derivation
- antonymy
- domain
- indomain
- codomain

This method is based on what are hereby referred to as *explorations* of the WordNet synsets graph, whose set of nodes will be here denoted with $\mathcal{S}$. Each exploration α starts from a set $I_\alpha \subseteq \mathcal{S}$ of synsets, each s having assigned a positive score $w_\alpha(s)$ of 1 or less: these are the first synsets to be explored. When exploring a synset s, for each s' of those related to it by certain pointer types and not previously explored, a score $r_\alpha(s') = r_\alpha(s) - \varepsilon_R$ is assigned, where $\varepsilon_R \geq 0$ is a *decay* parameter: synsets whose this score is greater than 0 will be explored in turn, the others are discarded. In the end, an exploration yields a set $E_\alpha \subseteq \mathcal{S}$ of explored nodes, each with a score given by r_α.

Given two input words w_a and w_b, the steps performed by the algorithm are the following. The first two steps are the same of the hypernyms-based

algorithm and make the algorithm early terminate if given conditions are met. In the final output vector, weight for each relationship type is assumed to be 0 if not assigned.

- If the two words are identical, terminate the algorithm by returning **equality** with weight 1.

- Extract sets S_a and S_b of synsets for w_a and w_b. If the two share at least one synset, return **synonymy** with weight 1. If at least one of the two sets is empty, return a null vector.

- Two explorations η_a and η_b starting from S_a and S_b with score 1 for all synsets are performed, each considering hypernymy and instance hypernymy pointers, with distinct decay parameters of ε_{hyp} and ε_{inst}. If in η_a has been explored at least one synset in S_b, set score for **hypernymy** to the maximum score within such synsets. Conversely, if in η_b at least one synset of S_a was found, do the same for **hyponymy**. Only if both these conditions fail, the intersection between the sets of explored nodes of η_a and η_b is checked: if contains at least one synset, consider the one s for which $r_{\eta_a}(s) \cdot r_{\eta_b}(s)$ is maximum and use this value as score for **coordination** relation, indicating that the two input words have a common hypernym.

- Similar steps are now taken with holonymy in place of hypernymy. Two explorations θ_a and θ_b considering all three types of holonymy pointers are started from S_a and S_b with all synsets having score 1; decay parameters for part, member and substance holonymy are respectively ε_{part}, ε_{memb}, ε_{subst}. If at least one synset explored in θ_a was also explored in η_b, **holonymy** relationship is assigned with weight equal to the maximum score of these synsets within θ_a. Symmetrically, weight **meronymy** as the highest score among synsets explored in θ_b which were also explored in η_a, if any. If neither of these two relationships hold, find the synset explored within both θ_a and θ_b for which the product of its scores given by the two is higher and, if any, use this product as weight for **co-occurrence** relationship, expressing in practice being part of a same entity.

- Explorations σ_a and σ_b are started from all synsets explored respectively within η_a and η_b (i.e. hypernyms of input words), from which

scores for initial synsets are taken ($\forall s \in I_{\sigma_x} : r_{\sigma_x}(s) = r_{\eta_x}(s)$): in these two explorations "verb group" and "similarity" pointers are followed, with respective decay parameters ε_{verbg} and ε_{sim}. These two relationships are considered together, as they both indicatively represent similarity between concepts, which are verbs in the first case and adjectives in the second. If one or more synsets are found in both explorations, excluding their initial synsets (for which relevant relationships have already been found), weight for **similarity** relationship is set to the product of σ_a and σ_b scores of the synset for which it is maximum.

- All couples of synsets explored in η_a and η_b are checked for lexical relationships of type "derivationally related form", "derived from adjective" and "participle" between words of synsets of one and words of synsets of the other. Weight for **derivation** relationship is set to the maximum product of weights of synsets across the two sets with words related by these relationships: this type of relationship generally links words with a common morphology.

- Similarly to above, the hypernym synsets are checked for antonymy relationships between words in a set and words of the other. Weight for **antonymy** is the maximum product of weights of synsets across E_{η_a} and E_{η_b} with antonym words.

- For last, "member of domain" relationships are considered. Explorations δ_a and δ_b starting from synsets explored in η_a and η_b are run, following all three types of such pointers (topic, region and usage) with a unique decay factor ε_{dom}. Similarly to hyponymy/meronymy, if one or more synsets are shared between E_{δ_a} and E_{η_b}, **domain** relationship is weighted according to the maximum score within δ_a of these synsets, while **indomain** relationship is weighted with the maximum score in δ_b of synsets also explored in η_a, if any. If none of these two relationships holds, **codomain** relationship weight is set to the maximum product of scores in δ_a and δ_b of non-initial synsets explored in both, if any.

The algorithm is parameterized by the decay factors referenced along the described steps and summarized in Table 5.2 along with their default values. These parameters have been tuned through some preliminary tests, training

Table 5.2 – *Decay parameters defined in the algorithm with their values*

Exploration	Pointer type	Decay parameters
hypernyms (η)	hypernym	$\varepsilon_{hyp} = 0.2$
	instance hypernym	$\varepsilon_{inst} = 0$
holonyms (θ)	part holonym	$\varepsilon_{part} = 0.2$
	member holonym	$\varepsilon_{memb} = 0.2$
	substance holonym	$\varepsilon_{subst} = 0.4$
similar synsets (σ)	verb group	$\varepsilon_{verbg} = 0.2$
	similar to	$\varepsilon_{sim} = 0.2$
domain of synset (δ)	domain (all 3 types)	$\varepsilon_{dom} = 0.4$

the search algorithm on lists of word pairs with pre-assigned relatedness or similarity scores, cited in §5.4.2. For each pair, we compared the reference scores given by these lists with the maximum value of the output vector from the search algorithm across a subset of relationships, having picked those deemed to best represent the intended concepts of "relatedness" and "similarity".

In general, the proposed algorithm has been designed to find various types of indirect relationships, even involving pointers of multiple types. The set of pointer types defined in WordNet and used in the previous algorithm is reduced to a smaller set of relevant relationship classes, grouping together similar types and defining new ones which can only be obtained by composition of primitive relationships.

5.5 Flat categorization

In this section, the general scheme of Section 5.3 is made concrete in a method for multi-label classification of documents within a flat taxonomy of categories. This is a very simple case, which requires few integrations in the scheme.

5.5.1 Example couples and category profiles

In flat categorization, the taxonomy where to organize documents is given by a simple set of categories $\mathcal{C}$. In this context, the possible relationships

between any document d and any category c are reduced to two cases: d is either labeled with c or not, according to some labeling $\mathcal{L} : \mathcal{D} \times \mathcal{C} \to \{0, 1\}$. In these two cases, the document and the category are said to be *related* or *unrelated*, respectively.

$$\mathcal{K} = \{\mathsf{related}, \mathsf{unrelated}\}$$

$$\Delta(d, c) = \begin{cases} \mathsf{related} & \text{if } \mathcal{L}(d, c) = 1 \\ \mathsf{unrelated} & \text{if } \mathcal{L}(d, c) = 0 \end{cases}$$

The set E of example couples extracted from the model training documents should be ideally composed equally of related and unrelated couples to yield a balanced training set for the semantic model. Anyway, it is typical that each document is labeled with few categories out of many possible ones, so unrelated pairs largely outnumber related ones. As a reference scheme for selection of training couples, it is assumed that for each training document $d \in \mathcal{D}_T$ a fixed number $n_{CpD} \leq |\mathcal{C}_T|$ of categories is selected randomly so that the number of related couples is the maximum possible. Specifically, if a document is labeled with more than n_{CpD} categories, then it is coupled with n_{CpD} of them picked randomly, otherwise it is coupled with all the related categories and with randomly picked unrelated ones to get to n_{CpD}.

Representation of each category in the flat case is built simply by averaging those for single documents labeled with it, whose set is denoted with $\mathcal{D}_c = \{d \in \mathcal{D}_T : \mathcal{L}(d, c) = 1\}$, as in nearest centroid classification. The average of standard vectors can be trivially adapted to the weighting functions used instead here as bags of words.

$$\omega_c(w) = \frac{1}{|\mathcal{D}_c|} \sum_{d \in \mathcal{D}_c} \omega_d(w)$$

5.5.2 Classification

As explained in the general scheme, the SMA is applied to the example couples to extract vectors of semantic features, which are then labeled as *related* or *unrelated* and used as training set for the semantic model, which is then combined with the same SMA to obtain a function predicting how much any document is likely to be related or not to any category. As there

Table 5.3 – *Summary information about benchmark datasets for flat categorization*

Dataset	Split	Docs. count	Docs. per category min.	avg.	max.
Reuters	training	9,603	182	719.4	2,877
multi-label, 10 categories	test	3,299	56	278.8	1,087
20NG	training	11,314	377	565.7	600
single-label, 20 categories	test	7,532	251	376.6	398

are two possible classes, the probability distribution returned by the model for a generic document-category couple will be of the form $(p, 1-p)$, with p being the likelihood with which the document should be labeled with the category.

In this setting, likely to standard method for multi-label flat categorization where specific classifiers are used, classification of a document d is performed by comparing it with profiles of all target categories to find which of them are related. Recalling what said above, for each category $c \in \mathcal{C}$, the semantic model returns a probability $p_R(d, c)$ for the *related* label. In general, by setting a threshold τ, can be established that a document is predicted to be related to all categories for which such probability reaches this threshold.

$$\hat{\mathcal{L}}(d, c) = 1 \Leftrightarrow p_R(d, c) \geq \tau$$

This yields for any document d a set of predicted categories, as entailed by multi-label classification. The method may also be easily adapted to the single-label case by labeling d with the category c with the highest relatedness likelihood.

$$\hat{\mathcal{L}}(d, c) = 1 \Leftrightarrow \operatorname*{argmax}_{\gamma \in \mathcal{C}} p_R(d, \gamma) = c$$

5.5.3 Experiment setup

The method has been tested on two datasets commonly used as benchmarks in text categorization research, already described in §3.8.1. Both of them are provided already split between a training and a test set, which are used

Table 5.4 – *Summary of parameters of the flat categorization method and of their default values (used where no different indication is given)*

Parameter function	Symbol and value(s)
Both training and classification	
Words considered by SMA for each document	$n_{WpD} = 30$
Words considered by SMA for each category	$n_{WpC} = 60$
Algorithm for search of semantic relationships	Θ varying
SRSM training	
Documents selected for example couples	$n_{ED} = 4000$
Example couples per selected document	$n_{CpD} = 4$

to train the semantic model and to test classification of documents using it as indicated in §5.3.6. Basic statistics about the two datasets are given in Table 5.3.

- The ModApté split of Reuters-21578 (9,603 training documents, 3,299 test documents) has been considered for multi-label classification. Likely to other works, classification under the 10 most recurring topics is considered. Namely, these topics are: EARN, ACQ, MONEY-FX, GRAIN, CRUDE, TRADE, INTEREST, SHIP, WHEAT, CORN. This dataset will be indicated in short with *Reuters*.

- The standard *bydate* distribution of 20 Newsgroups (11,314 training documents, 7,532 test documents) has been considered for single-label classification: every documents belongs to one and only one of the 20 groups. This dataset will be indicated in short with *20NG*.

The domains of the two datasets are somehow different and disjoint: while 20NG can be considered more general as it contains categories dealing with a variety topics (sports, motors, religion, ...), Reuters is instead focused on economy and markets, a discourse area which has not significantly representative categories in 20NG.

As seen in the description, the method has a number of parameters influencing its performances. Most tests are run by fixing some parameters to arbitrary default values and testing different values for remaining ones. Hereafter are described the parameters along with their default values, also summarized in Table 5.4.

Two important parameters of the process are the numbers n_{WpD} and n_{WpC} of most relevant words considered by the SMA respectively for documents and categories when comparing them. The number must be high enough to guarantee that all important words are considered for each document and category, but the number of queries to the semantic knowledge grows linearly with these numbers.

In the same context, is also relevant how the semantic relationships are computed. As discussed above, WordNet is always used here as a primitive semantic knowledge base, but different algorithms can be used to infer more high-level information from it. In the following, three algorithms are considered:

- the *direct* algorithm only searches and reports direct, primitive relationships, using the trivial techniques reported in §5.4.1;
- the *hypernyms* algorithm also searches for relationships between hypernyms of both words up to a number M_H of levels, as discussed in §5.4.3;
- the *extended* algorithm extends search through more types of pointers and returns fuzzy relationships, as discussed in §5.4.4.

For the hypernyms algorithm, $M_H = 2$ is assumed by default.

To train the SRSM, numbers n_{ED} of documents to be coupled to categories and n_{CpD} of categories to couple each document with must be set. To have a consistent training set, default values are set to $n_{ED} = 4000$ and $n_{CpD} = 4$, yielding a total of 16,000 example couples. As a learning algorithm for the SRSM, multivariate logistic regression is used [61].

As classification accuracy measures, commonly to most other works, the F_1 measure and the break-even point are reported for Reuters, whose calculation is discussed in §3.8.2. In the single-label case of 20NG, the classifier is required to report a single category, namely that with the highest similarity score for each document: here, accuracy (ratio of correctly labeled test documents) is reported instead as an accuracy measure.

5.5.4 Experiment results

In the beginning, to test the performances of the approach as a standard text categorization method, results are given for experiments where the

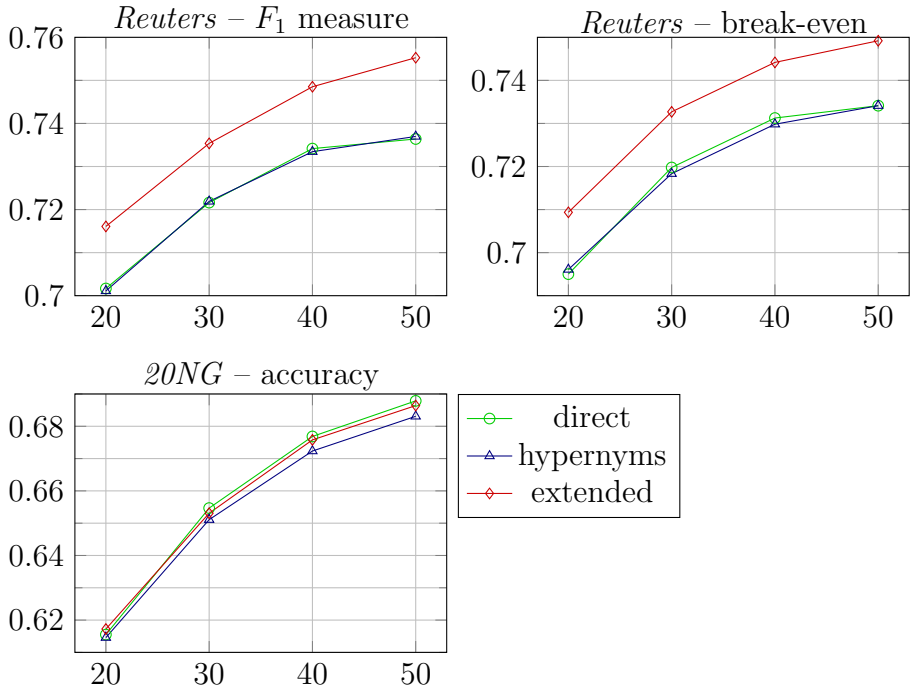

Figure 5.4 – *Accuracy measures (Y axes) for flat categorization as the number n_{WpD} of words considered for each document varies (X axes)*

corpus used to train the semantic model is the same on which it is tested subsequently.

Experiments shown first are those based on variation of the numbers n_{WpD} and n_{WpC} of words considered by the SMA for each document and category, respectively: different values are tested with the three different algorithms for searching semantic relationships listed above.

In Figure 5.4, results are shown for different values of n_{WpD}, where n_{WpC} is always set to the default value 60. The expectable result is that the accuracy measures grow as the number of words considered for each document is raised, as documents are better represented when comparing them to category profiles. The raise of accuracy for this range of values is of about 4% for Reuters and 7% for 20NG, showing that the contribute of "averagely" important words to represent the documents is at least in part helpful. In Reuters, the use of the extended algorithm appears to give an

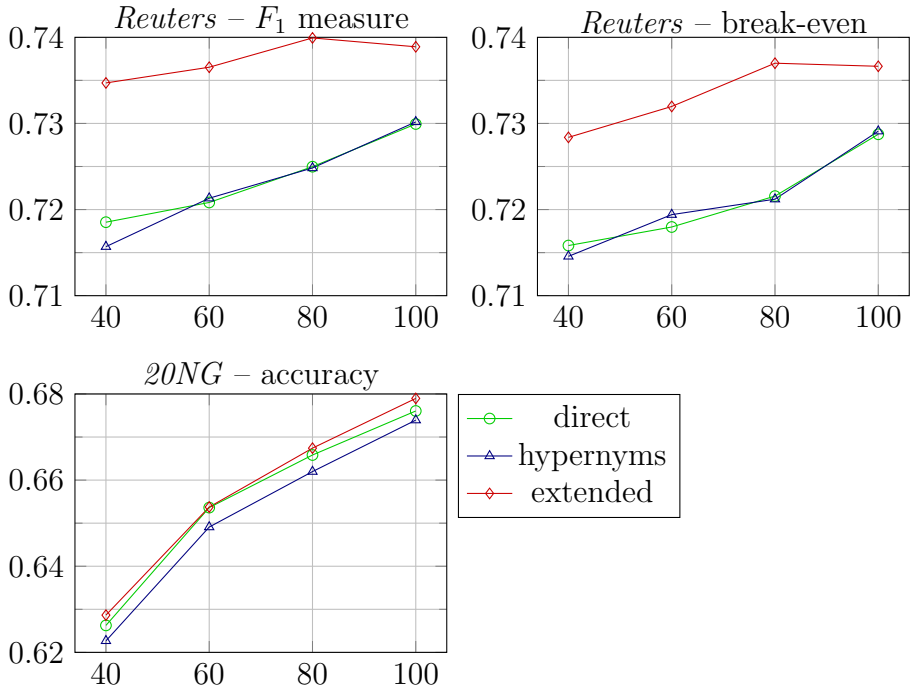

Figure 5.5 – *Accuracy measures (Y axes) for flat categorization as the number n_{WpC} of words considered for each category varies (X axes)*

accuracy boost of about 2%, but for the other cases the differences between the three algorithms are not significant.

Figure 5.5 shows the same results as n_{WpC} varies instead, with n_{WpD} fixed to 30. These results are essentially similar to those above: the use of more words to represent categories grants an increase in the accuracy, although with a slightly smaller difference in the considered range (1.5% for Reuters, 5% for 20NG). The observations above about semantic search algorithms still hold, with the extended one working better on Reuters.

Regarding the search algorithms, can be interesting to see what happens when their parameters are varied. Tests have been performed on the hypernyms-based algorithm, which only exhibits one parameter: the number M_H of levels of hypernyms to explore for each word. Figure 5.6 shows how accuracy on flat classification with this semantic search algorithm varies as its parameter is changed. Different results emerge for the two datasets:

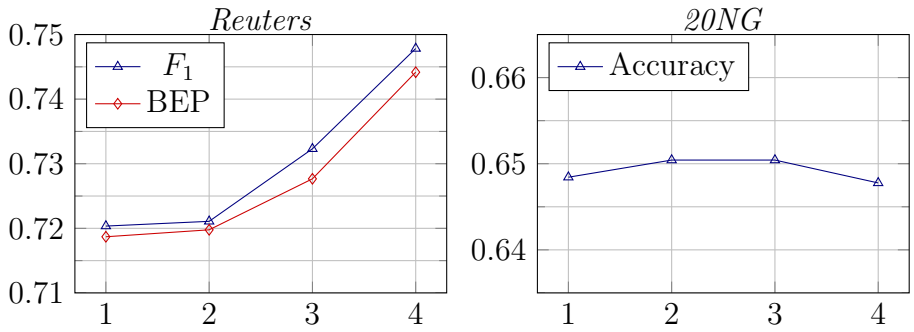

Figure 5.6 – *Accuracy measures (Y axes) for flat categorization using the hypernyms-based algorithm to search for semantic relationships as the number M_H of maximum considered hypernyms varies (X axes)*

for Reuters a deeper search makes some noticeable difference in the results, while for 20NG the variation of accuracy is minimal. It is supposed that the Reuters dataset contains many words with indirect hypernymy/hyponymy relationships, which require a proper algorithm to be found: this happens with the hypernyms based algorithm with a sufficient value for M_H and, as seen in the results above, also with the extended algorithm with its default values (given in §5.4.4), which likely grant thorough exploration of hypernyms.

In the experiments above, default fixed values for n_{ED} and n_{CpD} are used. To see how the composition of the training set of example couples for the SRSM influences the accuracy, different values are tested. Specifically, some tests have been performed by keeping constant the total number of example couples $|E| = n_{ED} \cdot n_{CpD}$ to 16,000, but doubling alternatively one factor while halving down the other. Additionally, tests where performed with selection of a reduced number of 2,000 example couples, given by $n_{ED} = 1000$ and $n_{CpD} = 2$ and also with selection of all possible example couples, so that $E = \mathcal{D}_T \times \mathcal{C}_T$, yielding 96,030 couples for Reuters and 226,280 for 20NG. Figure 5.7 reports the accuracy measures for all these tests. In the case of Reuters small differences emerge between the considered cases: the loss of accuracy using 2,000 couples instead of 16,000 is within 1%. On the other hand, with 20NG the differences are more evident, although they are much reduced when the extended semantic search

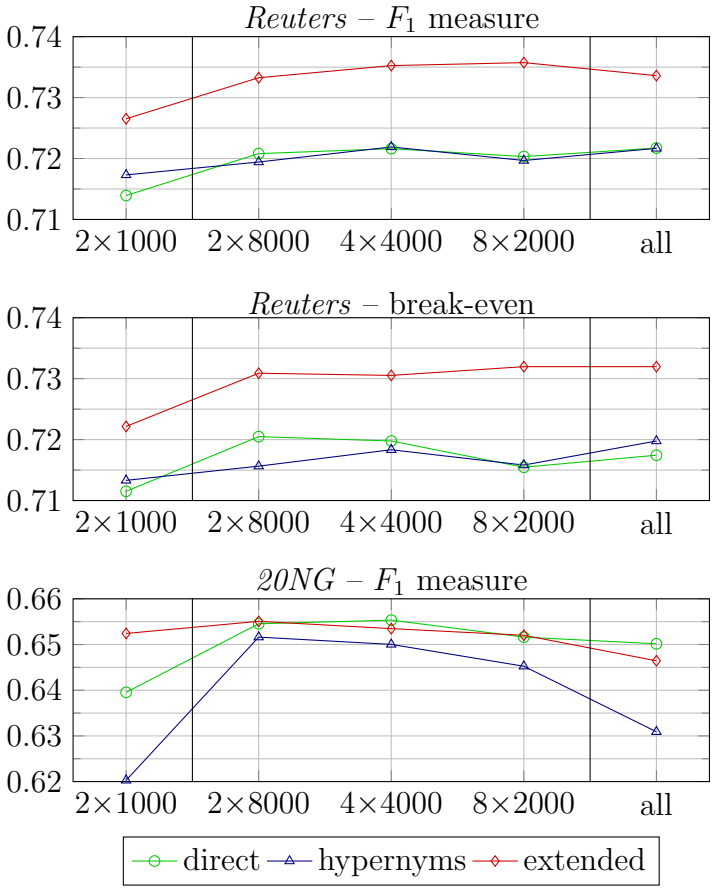

Figure 5.7 – *Accuracy measures (Y axes) for flat categorization as the composition of the set of example couples to train the SRSM varies: X axes labels are given in the form* $n_{CpD} \times n_{ED}$

Table 5.5 – *Comparison of results on flat categorization with those reported in other works*

Reuters – Comparison by break-even point	
this method (std. parameters, extended search)	0.733
this method (best result among tests)	0.749
Rocchio method [53]	0.799
Find Similar (Rocchio) [34]	0.646
SVM (best result with tested kernels) [53]	0.865
SVM [34]	0.920
20NG – Comparison by accuracy	
this method (std. parameters, direct search)	0.655
this method (best result among tests)	0.688
Rocchio method, *tf.idf* [52]	0.823
probabilistic *tf.idf* classifier [52]	0.903

algorithm is used. It can be noted that accuracy slightly tends to drop as matching documents to more categories is favored against considering more documents: this might be due to the fact that, when documents are matched to many categories, the training set for the SRSM tends to be highly unbalanced in favor of unrelated couples and the learning algorithm may be sensitive to this.

In comparison to the state of the art in flat text categorization, results given so far are not satisfactory. Table 5.5 shows a comparison among results obtained here with default parameters and other reported in relevant papers. Reported results for Reuters are those from [53] and [34] for the nearest centroid classifier (§3.6, referred to as Rocchio method) as the proposed method is somewhat an extension of it and for standard machine learning-based classification with support vector machines (§3.5.2), as the best overall performing method. Accuracy of the proposed method is about 5% below the best reported with nearest centroid classification and almost 20% below with respect to SVM. For what regards 20NG, comparison is done with results from [52] regarding the use of centroid-based classification and the probabilistic approach proposed there, which is among the best accuracy scores for this dataset. Also in this case, accuracy obtained here is inferior.

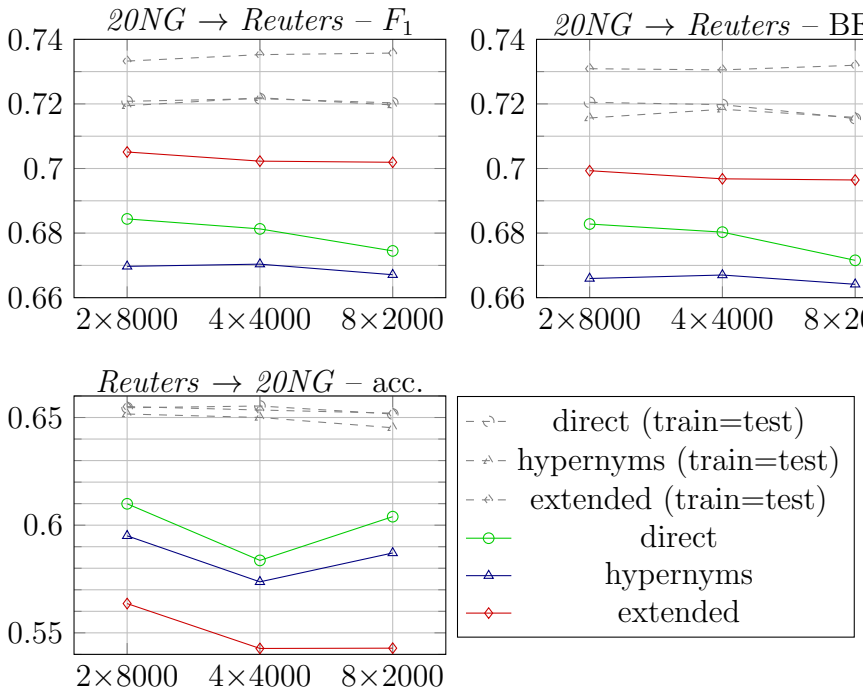

Figure 5.8 – *Accuracy measures (Y axes) for flat categorization with different SRSM training and classification test collections as the composition of the set of example couples to train the SRSM varies: plot titles are in the form "training corpus → test corpus", X axes labels are given in the form $n_{CpD} \times n_{ED}$; results which would be obtained by training the SRSM on the test collection are reported in gray*

All tests discussed above are performed each on the same dataset on which the SRSM is trained. To evaluate the actual independence of the semantic model from the specific domain upon which it is trained, it must be applied to a different domain. Following tests show the accuracy obtained by the method when classifying documents of a dataset by using a SRSM trained on a different dataset. The approach allows this even when, like as in this case, the datasets deal with different topics.

Figure 5.8 shows the accuracy obtained by crossing Reuters and 20NG as training and test collections – i.e. by using the SRSM model extracted with one to classify documents within the other – as the composition of

Table 5.6 – *Accuracy measures for possible combinations of training and test collections*

SRSM training collection	**Reuters** (extended search)				**20NG** (direct)	
	F_1 **measure**		**Break-even**		**Accuracy**	
Reuters (2×8000)	**0.733**		**0.731**		0.610	(-0.045)
20NG (2×8000)	0.705	(-0.028)	0.699	(-0.032)	**0.655**	

example couples varies. Results for each test collection are compared with those of Figure 5.7 where the SRSM is trained on the same collection. In the transfer of knowledge from one collection to the other there is some loss of accuracy, but in some cases it is of few percentage points. The worst cases are those applying a model extracted from Reuters on 20NG, where loss of accuracy can reach about 10%; on the other hand, when applying a 20NG-based model on Reuters using the extended semantic algorithm, the loss is as little as about 3%.

Finally, Table 5.6 summarizes the accuracy measures obtained for possible combinations of the two datasets used for training and evaluation, with the best tested combinations of parameters related to selection of example couples ($n_{ED} = 8000$ and $n_{CpD} = 2$ in all cases) and search of semantic relationships (extended algorithm when testing on Reuters, direct when testing on 20NG). For each cross-collection case, the loss of accuracy with respect to a SRSM extracted from the test collection itself is reported: in these cases with optimal parameters, the difference is within 5%, showing a not perfect but fair capability of the SRSM to obtain a generally valid model.

Between the two cases of transfer of knowledge across domain, the use of the model from Reuters on 20NG seems more difficult than the opposite, given the higher accuracy loss with respect to the standard case. This could possibly be given to the generation of a not fully accurate SRSM from Reuters, for which we hypothesize as a possible cause the slightly greater specificity of the domain, focused on economy, with respect to the general one of 20NG.

Summing up, experiments on flat categorization show that the method needs improvements to get a classification accuracy comparable to the state of the art, but they also suggest that SRSM models are already domain-

independent to some extent and can be applied in a context different from that where they are trained with limited penalties in accuracy.

About running times of the test, considering the whole process including model training and classification of test documents, they have been in a range between about 3 minutes and 3 hours, depending from the dataset and the parameters. The required time to use the hypernyms-based search algorithm is roughly triple with respect to that required with the direct algorithm; the extended algorithm requires in turn about triple the time of the hypernyms-based one. For other parameters, running time varies linearly with them. Classification on Reuters is about four times faster than on 20NG, considering the lower number of both documents and categories.

Experiments with hierarchical classification, presented below after the description of the method, are in part similar to those shown here for flat categorization.

5.6 Hierarchical categorization

While the previous section described how to apply the general method based on the domain-independent semantic model to flat categorization, here single-label classification in a hierarchy of categories is discussed instead. This setting entails a bit more complex implementation of the method, especially in the classification phase, for which a top-down approach is followed, similarly to methods with local classifiers described in §3.7.2.

5.6.1 Couple labeling and selection

Reusing the notation given in §3.2.2, the organization of categories of a set $\mathcal{C}$ in the hierarchy is defined by a relation $\prec$, so that $c_d \prec c_a$ indicates that c_a is an ancestor of c_d, i.e. a more general category. Considering single-label classification, each document d is associated with a single category, indicated with $l(d)$. As in the flat case, a document d is considered to be strictly related to a category c if $l(d) = c$, but the hierarchy of categories introduces also the possibility for $l(d)$ and c to be related within it.

Considering this, the *specialized* relation label is introduced for document-category couples (d, c) where the category is a generalization of the topic of the document, i.e. $l(d) \prec c$. While a *generalized* label would make sense for couples where the document topic is more general than the compared

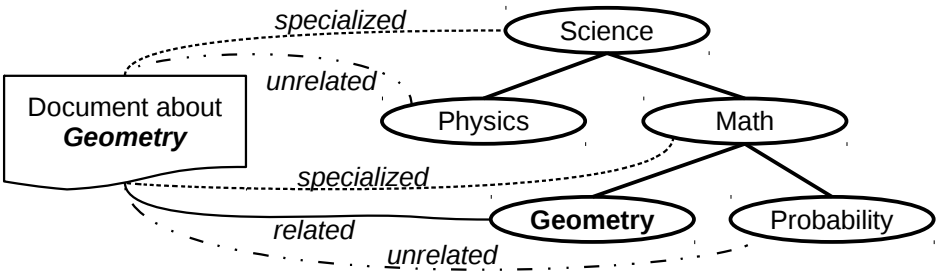

Figure 5.9 – *Coupling of a document with various categories in a hierarchy*

category ($c \prec l(d)$), it is not considered here, as in the top-down classification algorithm presented in the following a document should not happen to be compared with a more specific category. Considering this, the set $\mathcal{K}$ of possible couple relation labels and the relative labeling function Δ are illustrated in Figure 5.9 and defined as follows.

$$\mathcal{K} = \{\mathsf{related}, \mathsf{specialized}, \mathsf{unrelated}\}$$

$$\Delta(d,c) = \begin{cases} \mathsf{related} & \text{if } l(d) = c \\ \mathsf{specialized} & \text{if } l(d) \prec c \\ \mathsf{unrelated} & \text{otherwise} \end{cases}$$

The set of example couples E must then contain samples of all the three relationship labels, with as much balance as possible between them. However, given the single-label setting, the number of *related* couples is limited to one for each training document and specialized couples are also limited, being for each training document in number equal to the depth of the document in the hierarchy. Set two parameters n_{SpD} and n_{UpD}, a number of couples $N_{CpD} = 1 + n_{SpD} + n_{UpD}$ is selected for each document $d \in \mathcal{D}_T$:

- the *related* couple $(d, l(d))$ with its own category,
- as most *specialized* couples as possible up to n_{SpD} with ancestor categories $((d,c) : l(d) \prec c)$,

- the necessary number of *unrelated* couples to arrive to N_{CpD} (at least n_{UpD}).

This yields a total of $|\mathcal{D}_T| \cdot (1 + n_{SpD} + n_{UpD})$ example couples. The parameters n_{SpD} and n_{UpD}, contrarily to others, only have influence on the training set of the model and not on the classification using it.

5.6.2 Representation of categories

In the flat case, each category is represented simply as an average of all representative documents, which are trivially identified as those labeled with it. For hierarchical classification, in general, could be useful to take into account also documents of categories which are hierarchically related to the represented one.

Specifically, as a top-down algorithm is used, a very specific category dealt by a document must be distinguishable from the possible others since first steps, where the correct top categories must be chosen to follow the path which leads to the final correct category. For this, the profile of a category should integrate information of documents of more specific categories, in order to make their documents recognizable at higher levels. However, taking all documents in the whole subtree of a category could entail a high computational effort even for simply computing their average.

The adopted solution is, set a limit z of levels to explore, each category c has a *bottom-up* representation ω_c^H which is constituted by the average of documents labeled either with c or with a descendant of it whose tree distance is within z. Denoting with $c_d \prec_n c_a$ that c_d is a descendant of c_a distant n from it (e.g. $c_d \prec_1 c_a$ denotes that it is a direct child), a bottom-up representation is built as follows.

$$\omega_c^H(w) = \frac{1}{|\mathcal{D}_c^H|} \sum_{d \in \mathcal{D}_c} \omega_d(w) \qquad \text{where } \mathcal{D}_c^H = \{d \in \mathcal{D}_T : l(d) \prec_n c, n \leq z\}$$

Representations of this type are used as input for the SMA during the model training phase. Instead, within the top-down algorithm during the classification phase, also the *simple* representation ω_c made up of strictly related documents only is considered.

5.6.3 Top-down classification algorithm

While in the flat case a document is classified by comparing it with profiles of all possible categories, for hierarchical categorization a top-down approach is used: the document is first compared to profiles of categories of higher levels to progressively narrow down the range of its topics, until the most likely exact category is reached. This allows to avoid comparing a document with all existing profiles, which could be a very high number in hierarchical classification.

The algorithm is iterative, acting in each step on a set $C \subset \mathcal{C}$ of *current* categories: at the beginning, C is the set containing only the root node of the taxonomy of categories.

At each step, the combination of SMA and SRSM is first used to compare the document d with the simple representation of each category in C: the probability for each $c \in C$ to be *related* to d constitutes its *score*. In the following, the document is compared this time with the bottom-up representation of each category which is child of one of those in C: the score of each of these categories is computed as the probability to be in either *related* or *specialized* relationship with the document.

If one of the categories in C has the highest score of all (among both current categories and children), the classification algorithm terminates by returning it as the predicted category for d. Otherwise, a new set C' is built by taking at most a predefined number $n_{CC} \geq 1$ of the examined children categories having scores not lower than the remaining ones and also not lower than a threshold τ. If the resulting set C' is not empty, a new iteration is run using it as the new set C of current categories; otherwise, as no category is deemed to be enough likely related to the document, the algorithm terminates leaving the document *unclassified*, which expresses too high uncertainty about its correct category.

The parameter n_{CC} controls in practice the maximum number of branches explored throughout the search: if it is 1 only one path is tested, but for higher values the search can run through multiple, arbitrarily distant paths. The threshold τ has instead to be tuned to find an optimal value: a too low value may allow too much erroneous path in the classification, while a too high value may yield a high number of unclassified documents.

Table 5.7 – *Summary information about benchmark datasets for hierarchical categorization*

	Yahoo!		***DMOZ***	
Level	**docs.**	**cats.**	**docs.**	**cats.**
root	0	1	0	1
1	98	6	350	21
2	349	27	1,563	81
3	454	35	2,703	85
4	-	-	1,163	32
5	-	-	57	2
total	**907**	**69**	**5,836**	**222**

5.6.4 Experiment setup

Experiments on the hierarchical variant of the method have been carried out on the two datasets extracted from web directories (see also §3.8.1) and used as benchmarks in [17][4].

- The *Yahoo!* dataset was extracted from the SCIENCE branch of Yahoo! Directory: it contains a total of 907 documents organized in 69 categories.
- The *DMOZ* dataset was extracted from the CONDITIONS AND DISEASES branch of the HEALTH top category of the Open Directory Project: it contains a total of 5,836 documents organized in 222 categories.

Likely to those used in flat categorization experiments, these two datasets represent different domains: while DMOZ deals with a specific branch of the HEALTH theme, topics in Yahoo! are related more generically to science. As in the Yahoo! dataset there is no branch dealing with health, the two datasets have in practice no specific topics in common.

For each of the two datasets, 2/3 of the documents are priorly selected at random to constitute the training split, while the remaining 1/3 constitute the test split: both splits are used in the experiments as explained in §5.3.6.

[4]Both datasets are available for download at `http://www.di.uniba.it/~malerba/software/webclass/WebClassIII.htm`

Table 5.8 – *Summary of parameters of the hierarchical categorization method and of their default values (used where no different indication is given)*

Parameter function	Symbol and value
Both training and classification	
Sublevels for bottom-up category profiles	$z = 2$
Words considered by SMA for each document	$n_{WpD} = 10$
Words considered by SMA for each category	$n_{WpC} = 200$
Algorithm for search of semantic relationships	Θ varying
SRSM training	
Specialized couples per example document	$n_{SpD} = 5$
Unrelated couples per example document	$n_{UpD} = 5$
Document classification	
Probability threshold for specialization	$\tau = 0.15$
Maximum attempted categories	$n_{CC} = 3$

Regarding parameters, some of those present in the flat version of the method are also present here, with the addition of those related to the presence of hierarchy: these are the number z of sublevels of the taxonomy when building the bottom-up representation of each category, the parameters n_{SpD} and n_{UpD} related to selection of example couples and the parameters τ and n_{CC} of the hierarchical classification algorithm. Default values, used where not otherwise specified, are given in Table 5.8.

In this hierarchical context, the number n_{WpC} of words considered for each category by the SMA is substantially raised with respect to the flat case in order to support the high number of words for higher-level categories, which must be also representative of respective subcategories within up to z sublevels. n_{WpD} is reduced to balance the additional workload.

To train the semantic model, is used here the *Random Forest* learner [11], based on an ensemble of decision trees (briefly discussed in §3.5.3).

As accuracy measures for the hierarchical case, standard accuracy has been chosen to check how often the classifier picks the exact category, while hierarchical F_1 measure (hF_1 for short, see §3.8.2) is reported to indicate how much severe generally are the errors.

5.6.5 Experiment results

As above, are first considered experiments where the semantic model is trained on the same dataset on which is tested. Firstly, is analyzed the effect of varying the maximum number z of sublevels of the taxonomy where to retrieve documents to build the bottom-up representations of categories. Intuitively, this number should be high enough so that words belonging to more deep categories also appear in profiles of corresponding higher-level categories, so that, during the classification phase, these categories are correctly chosen. Results shown in Figure 5.10 confirm this intuition: for $z < 2$, a consistent loss of accuracy is experienced. Regarding efficiency, changing z does not have noticeable effects on the whole running time, as higher values generally just involve computing average of more documents for each category, which is a relatively efficient operation anyway.

Now, as done in the flat case, the effect of varying the number of words considered by the SMA for documents and categories is considered, along with the specific algorithm used to search for semantic relationships. In Figure 5.11 is shown how accuracy varies with the number of relevant words for each document, while in Figure 5.12 it is shown how it varies with the number of words per category. In both cases, obviously, the use of an excessively low number of words to consider has negative effects on accuracy. However, this effects are in some cases reduced with the use of more complex algorithms for finding semantic relationships, although the hypernyms algorithm appears to be not positively effective on the Yahoo! dataset.

Another parameter which can influence the performances is the number n_{CC} of open branches during top-down classification: Figure 5.13 shows the accuracy measures for different values of this parameter. It appears that in most cases it is necessary to explore at least two branches to obtain optimal accuracy, which instead drops when using only one and, to some extent, also when using too many branches, which in addition would make running times longer.

Is now investigated the effect of the number of example couples to train the SRSM on the final accuracy. Figure 5.14 reports the results for different values assigned to both n_{SpD} and n_{UpD}, indicating the number of specialized and unrelated couples produced for each document, in addition to the related couple with the document's own category. It is shown that these parameters can make a significant difference in the results, suggesting how

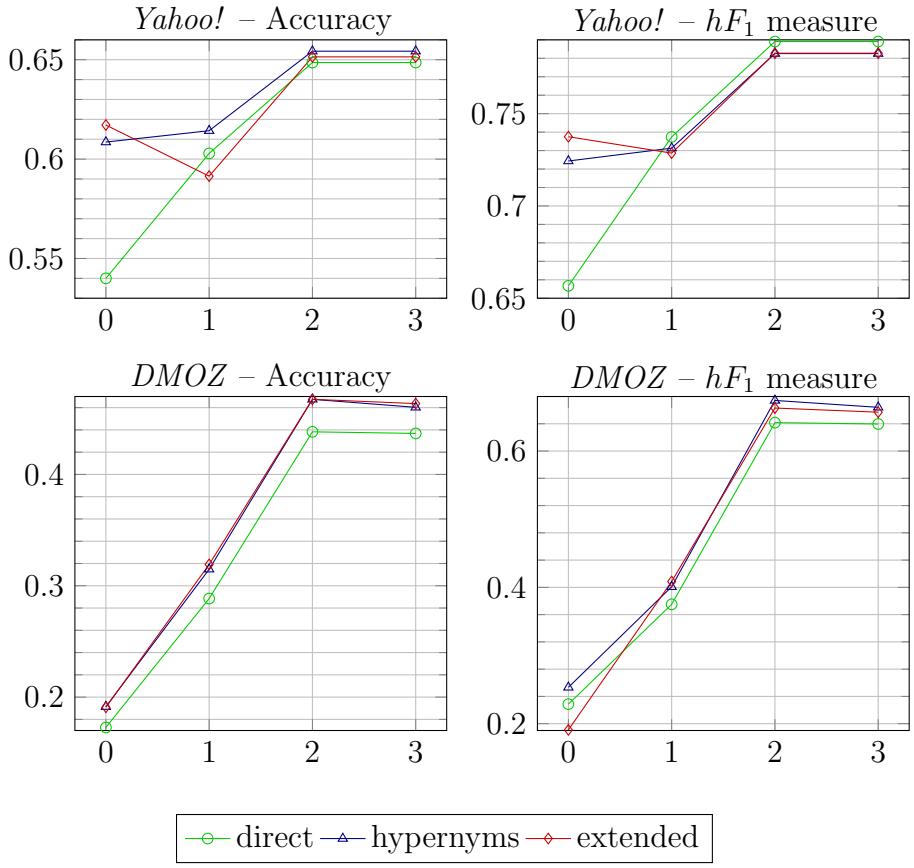

Figure 5.10 – *Accuracy measures (Y axes) for hierarchical categorization as the number z of sublevels of the categories taxonomy to consider when building bottom-up representations varies (X axes)*

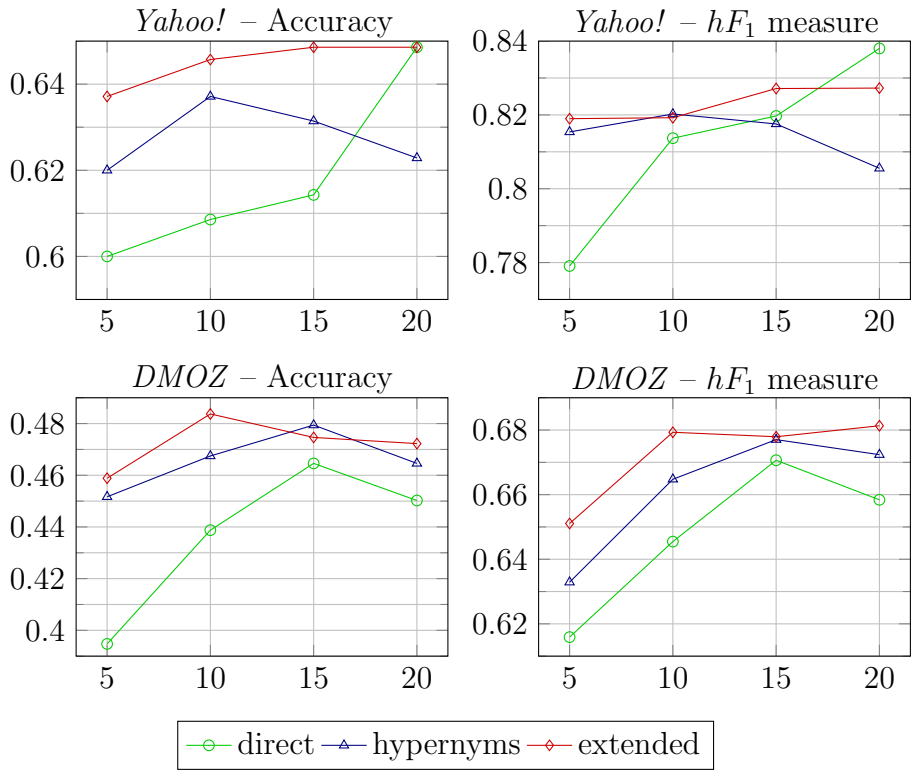

Figure 5.11 – *Accuracy measures (Y axes) for hierarchical categorization as the number n_{WpD} of words considered for each document varies (X axes)*

consistent must be the set of example couples to obtain an accurate enough model.

The obtained results, compared with the best ones reported in [17] for various methods and for the same datasets (although obtained with cross-validation rather than with a single training/test split) are partly satisfactory. This method obtains overall superior results in Yahoo!, while in DMOZ it does not outscore flat approaches applied to the dataset, but is superior to hierarchical ones.

Likely to the flat case, experiments where the semantic model is trained on a different collection from that of documents to be classified are considered, varying the semantic search algorithm and the number of selected example couples as above. Figure 5.15 shows these results, compared with

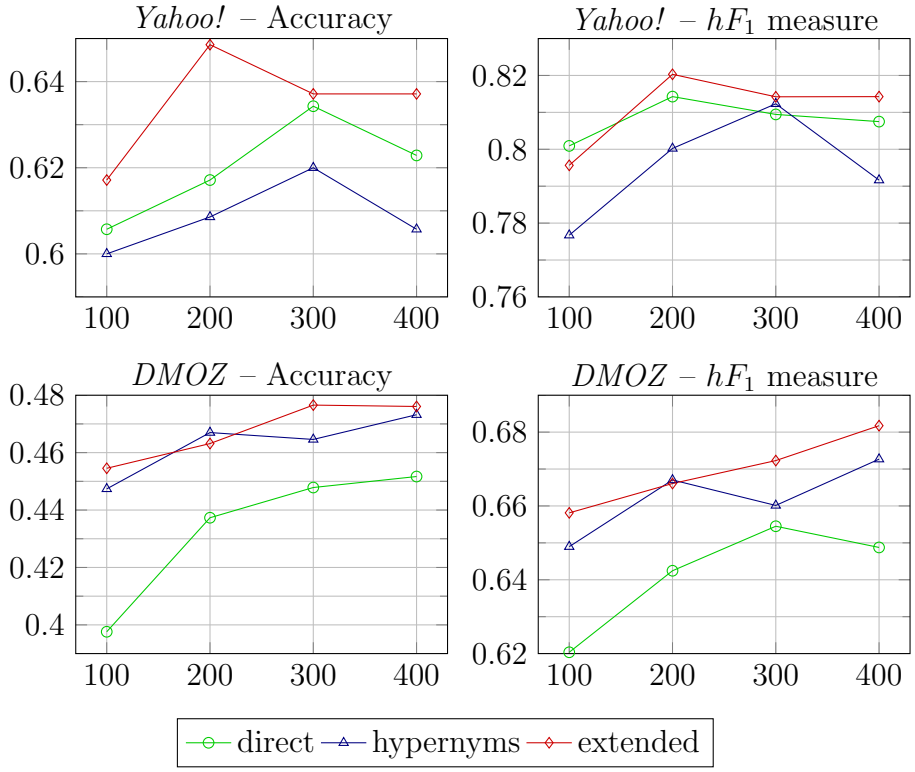

Figure 5.12 – *Accuracy measures (Y axes) for hierarchical categorization as the number n_{WpC} of words considered for each category varies (X axes)*

Table 5.9 – *Comparison of accuracy on hierarchical categorization with results reported by [17]*

	Yahoo!	**DMOZ**
this method (std. parameters, extended search)	0.6486	0.4837
this method (best results among tests)	0.6857	0.5053
flat, centroids-based	≈0.62	≈0.50
flat, SVM	≈0.61	≈0.61
hierarchical, naïve Bayes	≈0.55	≈0.41

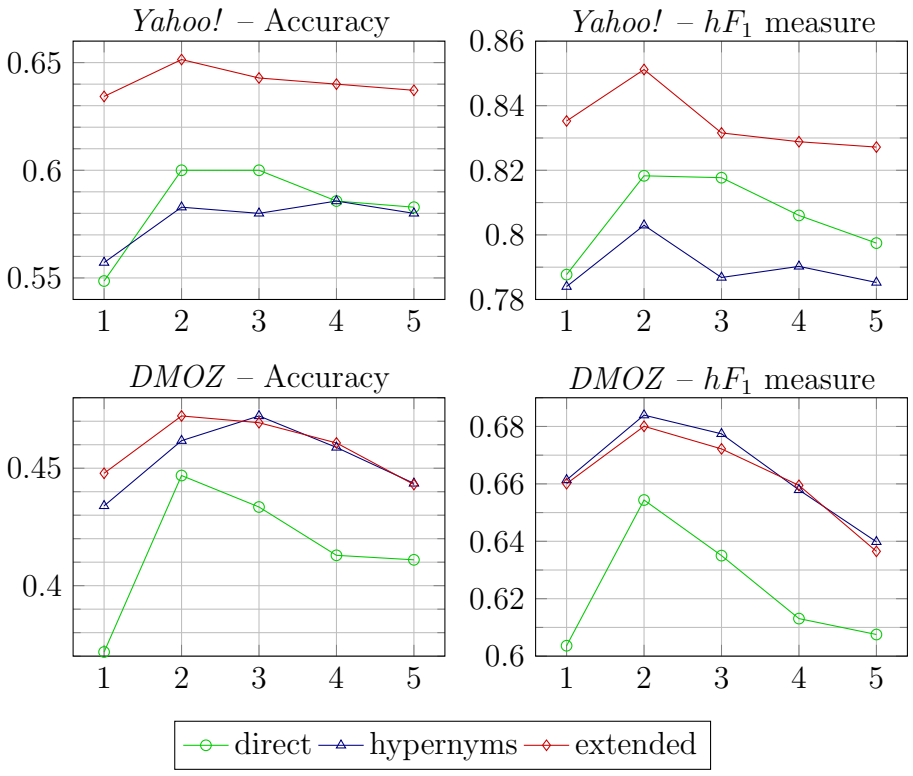

Figure 5.13 – *Accuracy measures (Y axes) for hierarchical categorization as the maximum number n_{CC} of categories tested at each level during classification varies (X axes)*

those already given in Figure 5.14 where training and test collections are not crossed. Among studied cases of transfer of knowledge across collections, the one where the SRSM trained on DMOZ is applied on Yahoo! seems to be the more problematic: the loss of accuracy with respect to using a SRSM trained on Yahoo! itself is generally within 10% and 15%, while hF_1 measure is about 10% below. The opposite setting, where the SRSM is trained on Yahoo! and used on DMOZ, gives a slightly smaller gap with respect to the base case: loss of both accuracy and hF_1 measure is generally within 10%, with cases where the difference is around just 2%.

Table 5.10 summarizes the accuracy and the hF_1 measure obtained for possible combinations of the two datasets with standard parameters and

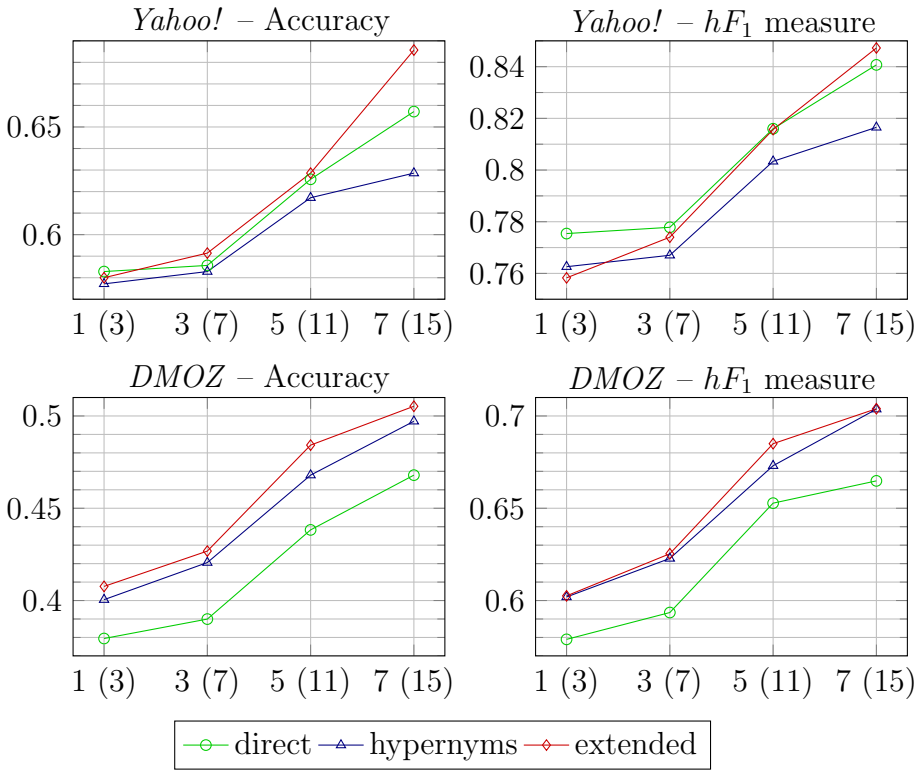

Figure 5.14 – *Accuracy measures (Y axes) for hierarchical categorization as the composition of the set of example couples to train the SRSM varies: X axes labels indicate the values of both n_{SpD} and n_{UpD} and between parentheses the total number of obtained example couples for each training document*

Table 5.10 – *Accuracy measures on hierarchical categorization for possible combinations of training and test collections*

SRSM training	**Yahoo!**				**DMOZ**			
	Accuracy		hF_1		**Accuracy**		hF_1	
Yahoo!	**0.629**		**0.816**		0.428	(-0.056)	0.666	(-0.019)
DMOZ	0.477	(-0.152)	0.705	(-0.111)	**0.484**		**0.685**	

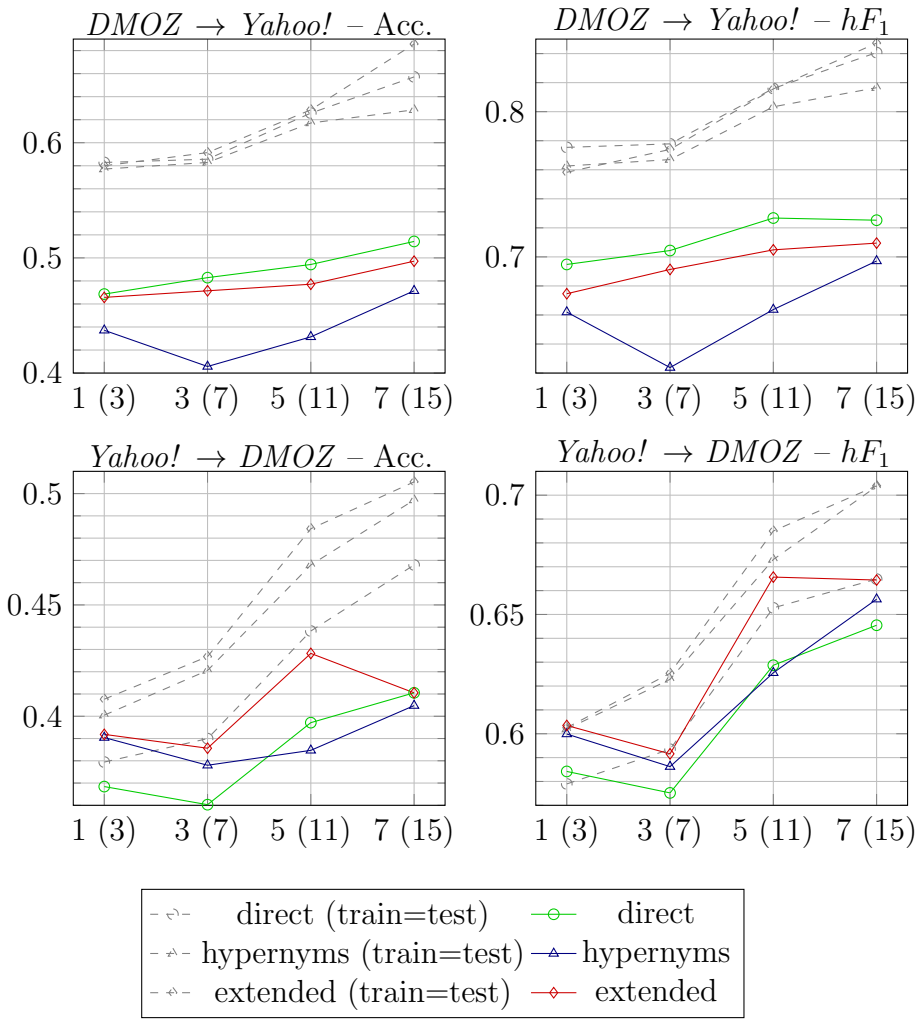

Figure 5.15 – *Accuracy measures (Y axes) for hierarchical categorization with different SRSM training and classification test collections as the composition of the set of example couples to train the SRSM varies: plot titles are in the form "training corpus → test corpus", X axes labels indicate the values of both n_{SpD} and n_{UpD} and between parentheses the total number of obtained example couples for each training document; results which would be obtained by training the SRSM on the test collection are reported in gray*

the extended semantic search algorithm. Again, for cross-collection cases, the loss with respect to the use of the same collection for training is shown. As from the plot above, from the table are visible the unsatisfactory result on classification on Yahoo! using a model from DMOZ and the instead interesting result with the opposite setting, with a quite small difference between the two models.

Regarding the relatively poor results on the DMOZ dataset, we hypothesise as one possible cause the specificity of its Conditions and Diseases domain. As WordNet is a general-purpose database without a relevant coverage of domain-specific knowledge, it is very likely that a significant number of important words found within the DMOZ dataset do not have a corresponding entry in WordNet, thus making impossible to identify semantic relationships involving them (apart from exact matching). The relative scarcity of word relationships potentially affects also the SRSM training phase and consequently the accuracy of the model: this could explain the poor results when using the model extracted from DMOZ to classify Yahoo! documents.

Most considerations about running times reported for flat categorization are also valid here: in these experiments, running times are roughly between 2 minutes and 3 hours, with experiments on DMOZ being about 3-4 times slower than equivalent ones on Yahoo!.

Summing up, the proposed method for hierarchical categorization, when the SRSM is trained on the same domain of documents to be classified, seems to yield good results in comparison with existing work on the same datasets. The noticeable difference between the "absolute" classification accuracy and the hF_1 measure in all cases suggests that at least some errors of the classifier happen in lower levels of the taxonomy. Regarding the capability of the SRSM to work across different domain, results are somewhat worse with respect to the flat case, with only one of two tested cases giving a fairly limited gap with respect to the base case of no transfer of knowledge.

5.7 Discussion

The proposed approach to text categorization is based on a knowledge model trained to predict the mutual relatedness between documents and categories by analyzing the semantic relationships between corresponding

representative words. The model trained on documents within a domain is structurally decoupled from its specific words and categories, potentially allowing it to be reused in different domains. Experiments were carried out to test the effective capability of extracted models to work across different domains as well as the general efficacy of the method in classifying documents, either in a flat or in a hierarchical set of categories.

Considering all the reported results on both the flat and the hierarchical variants of the proposed method, the results are partly interesting but still not fully satisfactory. Regarding the overall accuracy in documents classification, results for flat categorization are substantially below best methods such as SVM-based classification and, with smaller gaps, below standard nearest centroid classification, which somehow constitutes the skeleton for this method. Gaps with existing results are more favorable in the case of hierarchical classification, where obtained results are comparable and in part superior to those reported for the same datasets, which are however less diffused with respect to those used for flat categorization.

Throughout the experiments, some alternative methods have been tested to find semantic relationships between words. These methods, rather than returning a single relatedness or similarity score as most existing ones to compare couple of words, give a vector of scores for different relationship classes, allowing to differentiate between them. In many cases, the use of a carefully developed algorithm to identify relationships not directly indicated by the WordNet database has resulted in improvements of few percentage points of the accuracy measures.

Regarding the presumed generality of the semantic model to predict document-category relationships, results of tests partially support this hypothesis. When classifying documents on the basis of a semantic model extracted from a different domain, the final accuracy is generally lower with respect to using a model obtained from the same domain, but in many cases the loss is limited to some percentage points. This happens despite of differences in the topics treated by the different test collections, suggesting that the obtained models are effectively domain-independent to some extent. Results seem to suggest that, to obtain a SRSM which works well across domains, it should be trained from a domain which is as more general as possible: infact, lower accuracy measures appear to occur exactly when the model is trained on domain-specific datasets.

By also looking at the results of other methods on the same datasets, we can suppose that the sub-optimal results are mostly a consequence the

centroid-based nature of the method. As discussed earlier and seen in the comparison of results, nearest centroid classification has been proven not to be among the best approaches to text categorization. We chose to use it for being the most straightforward method to have self-contained profiles of categories resembling bags of words for documents, so that each category is represented by a list of weighted relevant words. Potential room for improvement could be in using finer models for representation of categories, which however requires either to fit them in the existing scheme involving the search of semantic relationships with relevant words of documents or to adapt somehow this scheme to accept sensibly different models.

Another aspect which could be improved is the search of semantic relationship. While the use of finer search algorithms helps in finding more useful relationships and often results in accuracy improvements, we anyway experienced, while testing such algorithms, that some intuitive relationships between words are still not found; it also happens that links are returned between actually unrelated words. These issues with false negatives and false positives could be possibly solved by further work on the semantic search or, probably better, from the use of more rich semantic knowledge bases. From the emerging Semantic Web, it is possible to extract knowledge about a large number of objects, with many meaningful links between them.

Other parts of the method could also be refined to attempt to improve the overall results of the method. Other than trying different values for parameters, which would anyway have in many cases negative effects on the running times, changes could be made in the creation of categories profiles (for example by considering negative examples, as happens in general in the Rocchio method), in the selection of example couples, in the search of semantic relationships or in the training of the semantic model. Additionally, tests could be performed on larger datasets, testing for example if more general semantic models, with optimal accuracy on other datasets, can be extracted from them.

Chapter 6

Conclusions

In the thesis, techniques for text mining, consisting in the automatic analysis of possibly large collections of text documents, have been discussed. The presentation of such techniques constitutes a general overview of the literature about text mining, borrowing from fields such as machine learning and information retrieval. These techniques mostly serve to create suitable structured representations of documents, which can be then handled by many algorithms.

Focus has been given to text categorization, a task prominently addressed within text mining with different applications, among which distinction of topics discussed within documents is the most recurring genre. In the common machine learning-based approach, a knowledge model able to classify new documents is extracted from a training set of already labeled ones, usually handled in form of bags of words.

The first contribution addressed cross-domain text categorization, where a knowledge model to classify documents in a target domain is obtained with the support of labeled documents of a source domain. A novel method has been proposed, based on nearest centroid classification, where a profile for each category is extracted from training documents and each new document is classified with the category having the closest profile. In the cross-domain case, such profiles are extracted from the source domain and, through successive refining steps, they are adapted to the target domain and then used to classify relevant documents.

Experiments on recurring benchmark datasets produced good results, with accuracy measure superior or comparable to those reported by many

works and fast running times, all despite the relative simplicity of the approach. A variant is proposed to cut down the number of iterations with minimal loss of accuracy, while another one is proposed to obtain from the source domain a regression model mapping measurable similarity to effective relatedness.

Starting from the simple idea of this model, a more complex one based on semantic relationships between words has been conceived: the idea is to deduce the degree and also the nature of relatedness between arbitrary documents and categories from the semantic relationships between the respective relevant words within them. The proposed model is trained from a set of labeled documents, but once built it is decoupled from them, as references no specific words or categories. It is theorized that the model is effectively domain-independent, able to accurately predict the relationships between documents and categories from their representations even outside of the domain it was trained into.

To test this assumption, a general scheme for text categorization based on this model has been proposed, which could grant advantages like the possibility to introduce new categories at any time while classifying documents without the need to train new or updated models, but just by providing profiles built efficiently from relevant documents. Two instances of this scheme have been proposed for flat and hierarchical categorization, respectively: in the latter case, a top-down algorithm is used to allow efficient classification within taxonomies with many categories.

Knowledge of semantic relationships between words is based on the lexical database WordNet, which stores primitive links between concepts. In order to identify indirect relationships which exist but are not directly given, two algorithms with different complexity have been proposed, which have been tested in categorization experiments along with the solution to simply look for direct relationships only.

Experiments on flat categorization have been carried out on two largely diffused datasets, testing different values for some key parameters. Accuracy measures are below those reported by other works on the same datasets, but tests showed that a semantic relatedness model extracted from one of the two dataset can be employed to classify documents within the other dataset with a limited loss of accuracy, namely within 5% in the analyzed cases.

Also concerning hierarchical categorization, two datasets have been considered for experiments. In this case, comparison of accuracy with reported

results for other methods on them is partly satisfactory. With these two datasets, applying a model extracted from one to classify documents in the other brings results about as good as above in one case and worse results in the other.

Overall, the experiments show that classifying documents using knowledge extracted from a different domain yields an accuracy which often is only few percentage points below that which would be obtained by using a model extracted from the same domain. This partly supports the hypothesis of the general validity of the semantic relatedness model. It is expected that some improvements could be made with further tuning of parameters and especially by using models extracted from larger datasets, possibly providing more rich knowledge.

Comparison of the different approaches to identification of semantic relationships show that the more complex algorithm generally guarantees slightly higher accuracy with respect to more simple methods, although with higher running times. Possible further work on these algorithms will be focused on carefully selecting the operations to be performed, in order to retain good efficacy while improving the efficiency.

6.1 Ideas for future research

A relevant limit of the presented work, concerning both the cross-domain categorization method and the semantic model, is the centroid-based nature, which is likely an important cause of the unsatisfactory accuracy measures in the latter part. Works on text categorization show that methods based on nearest centroid classification usually bring sub-optimal efficacy due to the fact that categories cannot usually be accurately represented with single centroids. We have chosen this representation as a straightforward solution to experimentally investigate the possibility to extract domain-independent knowledge. Future work might be dedicated to the search of a more robust model for representation of categories, to be fit in the proposed approach.

For what regards the concrete text categorization method based on the domain-independent semantic model, further research may include, other than experiments on larger datasets and with different values of parameters, test of variants in different parts of the method, such as the selection of example couples and the classification process. The goal is to obtain

improvements on the generality of the obtained models and on the overall accuracy of classification of documents.

Other possible directions include the application of the proposed approach to different tasks, such as performing clustering of documents, by using the semantic model to measure mutual distances and relationships between documents. The distinction of multiple relatedness modalities, like distinguishing strictly related documents from those treating more general or more specific topics, may be useful in more advanced applications, such as hierarchical clustering and induction of a taxonomy of categories.

Appendix A

Network Security through Distributed Data Clustering

While the rest of the thesis has its focus on text mining and specifically on text categorization, here is presented an additional work developed during the Ph.D. course, consisting in the design of a system for detection and identification of malicious traffic within a computer network based on distributed data mining. In this system, nodes of a network continuously gather statistics on the traffic obtained from SNMP agents on monitored hosts and periodically run a distributed data clustering algorithm to cooperatively build a knowledge model able to separate observations of normal traffic from those relative to different types of attacks.

A.1 Distributed data mining

As described in §1.1.2, machine learning techniques are generally within data mining used to extract knowledge models from potentially large volumes of data, which can be used to formulate predictions on subsequently produced data.

A branch of machine learning which is gathering interest in present times is that of *distributed* learning algorithms, which run on multiple separated machines with more or less limited capabilities to communicate among them. A "distributed algorithm" in general is distinguished for the distribution of *computation* across machines, in order to split the load of work

and thus reduce the total time necessary to complete it. In the context of data mining and machine learning, it is often considered to have *data* distributed as well: each machine has a different set of observations and must cooperate with the others to produce coherent general models based on their distribute knowledge.

These algorithms are useful in networks of nodes where each of them gathers significant amounts of data which should be analyzed collectively, but transferring all data to a central point would be costly, inefficient or unfeasible for other reasons, such as for privacy concerns between different parties. Many distributed learning algorithms work with minimal exchange of information: nodes are connected according to a network topology and exchange small messages with neighbor nodes only, thus limiting costs for their interconnection and latency of communications [22].

A.2 Network intrusion detection systems

Information security is an essential issue in distributed systems: any host connected to the Internet is a potential target of a variety of network-based attacks trying to discover possible system vulnerabilities (by means of, e.g., port scanning) and to exploit them for malicious intents, for example making network services unavailable to legitimate users (e.g. Denial of Service attacks) or to gain unauthorized access to them (e.g. brute force or buffer overflow attacks). To contrast these network threats, specific countermeasures have been devised and already deployed in production environments, while the research community is still working to improve them.

Network-based Intrusion Detection Systems (NIDSs) have been developed to automatically detect potential attacks coming from the network, report them to the system administrators and, in some cases, even try to prevent them. The general model of intrusion detection systems, also used to monitor hardware and software on single hosts, was introduced in [31]: from this general idea, various solutions for network security have been proposed [54, 71].

Two major approaches for intrusion detection systems can be distinguished. *Signature-based* systems require the preliminary knowledge of some sort of behavioral patterns of the threats to be detected, so that the system can trigger an alert as it detects a signature of a specific attack in

the observed traffic. This type of IDSs has the natural limitation of being able to detect only known threats, which can be described at different levels of detail. On the other hand, *anomaly-based* IDSs are trained to recognize the normal behavior of the monitored system and their task is to report any significant divergence from it. These systems are more likely to detect previously unknown types of attack, but they need a precise and often complex definition of the correct behavior of the system.

Some works proposed the use of machine learning techniques for intrusion detection, with [62] being the first to apply supervised learning techniques to network security, while the use of unsupervised learning started from [98] with the use of single-linkage clustering. These methods are based on analysis of raw traffic or summarized data obtained from platform-specific tools (e.g. Unix' `netstat`).

Some newer methods leverage instead information obtainable from the SNMP (Simple Network Management Protocol) agents usually running on many hosts, which gather statistical information about network activity, stored in a Management Information Base (MIB) with a standard structure: these methods are then more easily portable across platforms. A first example of MIB-based NIDS is given in [13] where abnormal rates of change of some variables are detected. An example of application of machine learning to MIB data is given in [127], where Support Vector Data Descriptions (SVDD, variant of one-class support vector machines) are used.

A.3 General system model

We present here the general architecture of the proposed system. It is composed by a set of *monitoring stations* that form a peer-to-peer (P2P) network according to a given overlay scheme: this way, each station has a number of neighbors it communicates with. Each station (or *node*) collects MIB data with statistical traffic information from a set of hosts (servers, workstations etc. and/or even itself) through the SNMP protocol, by querying them at regular time intervals.

Each node gathers then a continuously growing set of *observations*, each containing part of the information given by the SNMP protocol about network traffic traversing a given host at a given time, constituting a concise snapshot of such traffic. At some moments during this data collection process (e.g., at regular intervals), all the nodes in the network start a

distributed data clustering algorithm, each using its own SNMP observations as a local data set. The objective of the data clustering algorithm is to partition the "global" data set (the union of all observations stored by all nodes in the network) into a number of clusters constituting groups of similar instances of data.

Each one of the detected clusters corresponds to a particular traffic situation, which may denote either a regular network activity or the presence of an attack against a host or the entire network it is part of. Initially, a node is not able to determine whether a given cluster corresponds to a normal situation or to an ongoing attack: therefore it is necessary to have some form of user supervision to match located clusters to traffic situations as long as these are unknown. However, after a node knows that a cluster corresponds to a particular network threat, it can use this information to detect that threat. While collecting observations from SNMP, a node can trigger a warning if some of these fall into a cluster which is known to correspond to an attack. By periodically repeating the execution of the distributed clustering algorithm over time, the knowledge of nodes is progressively updated and previously unknown types of attacks can be identified.

A.4 Implementation details

The previous section describes the general structure of the system, leaving much freedom about its design and implementation. We illustrate here some details about a possible implementation we used as a reference, used for the simulations described subsequently.

One thing to determine is the set of features of which each observation extracted by monitoring stations is composed: SNMP provides a large amount of information, but processing all of it can be inefficient. This issue can be addressed by means of feature selection, using specific techniques to select a subset of all the possible features which can be much smaller but can represent the observations with minimal loss of information. In our simulations (see below), once extracted the test dataset, we run on it the *correlation-based feature selection* algorithm [45], which selects attributes poorly correlated with each other to avoid redundancy: the algorithm has been run on multiple folds of the dataset, then the 14 features selected in at least half of the folds have been picked. In a real setting, this feature se-

$D^n \leftarrow$ local data set
$k \leftarrow$ number of clusters (same for all nodes)
$\gamma \leftarrow$ termination threshold (same for all nodes)
$M_1 \leftarrow$ initial centroids (same set for all nodes)
$i \leftarrow 1$
repeat
  $A_1, A_2, \ldots, A_k \leftarrow \emptyset$
  for all $d \in D^n$ **do**
    $A_x \leftarrow A_x \cup \{d\} : x = \arg\max_{x \in \{1,\ldots,k\}} (\left\| d - m^n_{i,x} \right\|)$
  end for
  for all $j \in \{1, 2, \ldots, k\}$ **do**
    $l^n_{i,j} \leftarrow \frac{1}{\#A_j} \sum_{d \in A_j} d$
  end for
  Send local centroids L^n_i and counts $c^n_{i,j}$ to neighbors $a \in Adj(n)$
  Receive centroids L^a_i and counts $c^a_{i,j}$ from neighbors $a \in Adj(n)$
  for all $j \in \{1, 2, \ldots, k\}$ **do**
    $m^n_{i+1,j} \leftarrow \frac{\sum_{a \in Adj'(n)} c^a_{i,j} \cdot l^a_{i,j}}{\sum_{a \in Adj'(n)} c^a_{i,j}}$
  end for
  $i \leftarrow i + 1$
until $\max_{j \in \{1,\ldots,k\}} \left\| m^n_{i,j} - m^n_{i-1,j} \right\| \leq \gamma$

Figure A.1 – *Pseudo-code for the Local Synchronization-Based P2P K-means, as executed by a node n.*

lection process could be run periodically to keep the optimal set of features up to date, although it should not need to be repeated as frequently as the clustering process.

As a concrete distributed clustering algorithm for experiments, an adaptation proposed in [27] of the classic *k-means* partitive algorithm has been used. All nodes start from a common set of k *center* points in the observations space, then iteratively each node n executes the standard k-means actions, followed by exchange of information with the peers $Adj(n)$ which are neighbor in the topology. First, as always, each observation is assigned to the nearest center and centroids of observations for each group are then computed. Subsequently, these obtained *local* centroids are sent to the

neighbors, which send in turn their ones: the node then computes the new position of each center as the average of the centroid positions in the node itself and in its neighbors ($Adj'(n) = \{n\} \cup Adj(n)$), weighted according to the number of observations assigned to it in each one. When, from one iteration to the other one, the position of every center is not moved for a distance greater than a threshold γ, the node has reached convergence and stops iterating, continuing anyway to communicate its local centroids to neighbors still running the algorithm. In the end, each node has a clustering model based on the collective knowledge in the network. Figure A.1 shows the pseudo-code for the algorithm.

A.5 Simulation setup

To test the potential effectiveness of the proposed system in distinguishing observations generated from normal traffic and those generated from different types of attacks, we have simulated the execution of the distributed k-means-based clustering algorithm described above on an artificially generated yet realistic set of SNMP observations.

We first generated the dataset by setting up a network with a "victim" server monitored by a separated host and connected to a network including machines simulating both regular clients and attackers. We gathered SNMP data at regular intervals from the monitored server in five different sessions. During the first session, we emulated regular network behavior by generating HTTP requests from the clients. Then, in the following four sessions, we added malicious traffic from the attackers by placing the same server under different kinds of network attacks. The attacks used in the respective sessions were:

1. Denial of Service,
2. Distributed Denial of Service,
3. Denial of Service on SSH,
4. Brute force on SSH.

We produced a total of 5,655 observations divided into five classes, according to the session each one was generated in, so one class corresponds to the absence of attacks, while each of the remaining four corresponds to one of the listed network attacks. After building this dataset, we reduced its initial set of hundreds of features to only 14 representative ones, using the selection algorithm mentioned above.

This dataset has been tested on multiple simulations of the clustering algorithm. Each simulation works by setting up a virtual network of nodes, assigning a training and a test set to each of them, running the algorithm with each node using its training set and finally measuring accuracy indicators on all nodes using their respective test sets. To assign data to nodes, we used specific *data distribution* algorithms to extract from the whole dataset a training subset and a test subset for each node. Simulations are different from each other for three principal aspects: the topology of the network on which the algorithm is run, the distribution of data across the nodes and the parameters specific to the clustering algorithm. Considering the distributed k-means variant presented above, we mainly tested the variation of the number k of clusters.

Five different topologies with 64 nodes have been tested: a *scale-free* network, a ring, a ring with 16 random additional links, a torus and a fully connected mesh. Regarding distribution of data across nodes, two different general strategies have been tested to provide a training and a test set to each node: (A) distributing different observations of the same n classes to the two sets or (B) picking independently for each n classes and provide all observations of them. In all cases, data is distributed independently to each node and the distributed classes always include that of regular traffic, as it is supposed to be observed much more than the others in a real case.

For each tested combination of parameters, 20 random distributions of data are considered from the picked distribution strategy; for each of these, the distributed k-means algorithm is run 50 times with different random initial positions of the centroids. From these 50 runs, the one with the best results is considered, assuming the application of existing methods for optimal centroid initialization; the results of the 20 "best-case" simulations with differently distributed data are then averaged, as the distribution of data cannot be controlled in a real case.

For each single simulation, all accuracy measures are averaged across all nodes. The main accuracy measure, used to determine which are the best runs, is the ratio of test observations correctly classified considering the best possible mapping from clusters to classes (assumed to be supervised), referred to as *attack identification* accuracy. Other than this, *attack detection* accuracy is also measured, which only considers the ability of the nodes to distinguish regular traffic from attacks, regardless of their class, so that detecting an attack of a type in response to one of a different type is not considered an error. At this extent, to have indications about the type

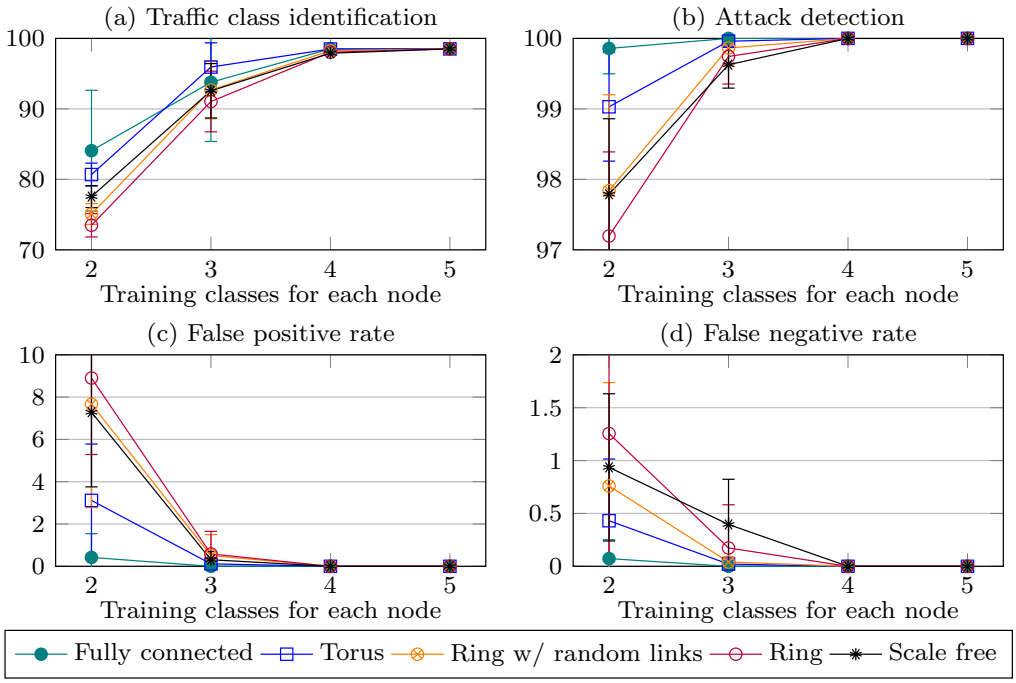

Figure A.2 – *Percentage accuracy measures for distributed k-means (on Y axis, in percentage) run on different 64-node topologies, using data distribution strategy A (train and test on different data of same classes) with a variable number of training classes (on X axis).*

of committed errors, *false positive rate* (ratio of regular traffic observations misclassified as threatening) and *false negative rate* (ratio of undetected attack observations) are also measured.

A.6 Simulation results

A parameter with an important influence on the results is the distribution of data across nodes, especially the number of classes known by each node. Considering both the strategies proposed above, if each node knows 4 classes or all 5 of them, the detection is perfect and the identification accuracy is generally above 98%; we found out these to be the same results which can be reached by running the standard centralized k-means

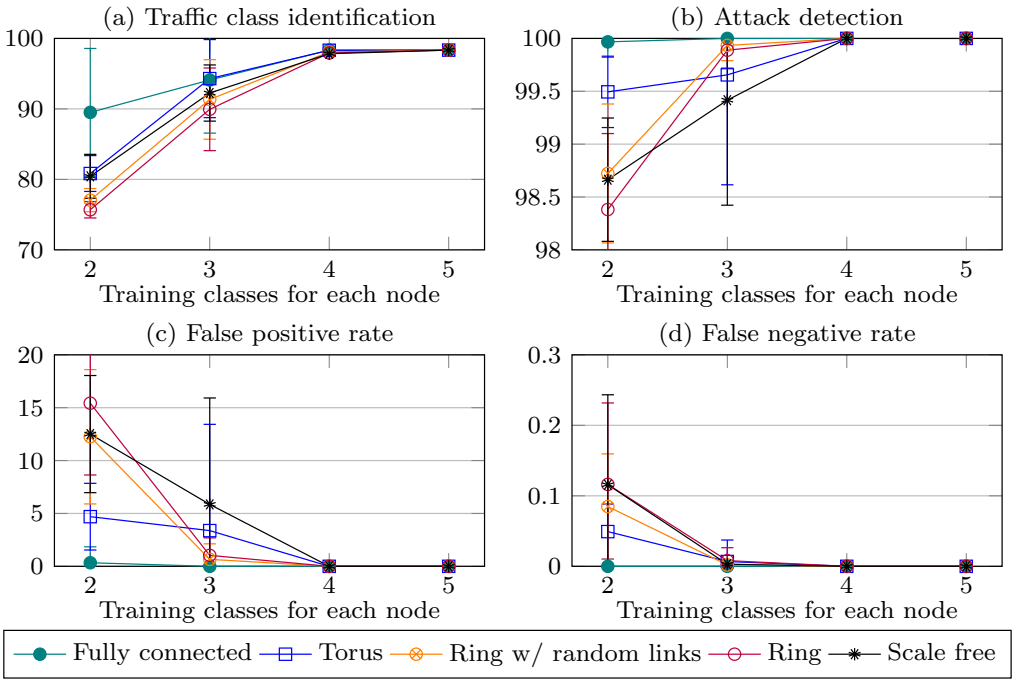

Figure A.3 – *Percentage accuracy measures for distributed k-means (on Y axis, in percentage) run on different 64-node topologies, using data distribution strategy B (train and test on different classes) with a variable number of training classes (on X axis).*

algorithm on the whole dataset. Instead, with 2 or 3 known classes, the detection is in most cases over 97% and 99% respectively, with false positives more common than false negatives, while identification is usually above 90% with 3 classes per node. Figures A.2 and A.3 show these results on all tested network topologies for data distribution strategies A (same classes for training and test set) and B (sets with different classes) respectively: as by what explained above, each point shows the average on 20 runs with different random distributions, with error bars indicating variance.

From the plots, can also be noted that using network topologies with higher connectivity degrees can boost accuracy, as each node is more likely to receive information about classes he misses. A comparison can be made for example between using a plain ring topology and one with 16 additional

links, for which results are given in the plots. Testing further quantities of additional links in the ring network, we experienced accuracy measures to grow roughly linearly with such quantity, in cases where they are not already at optimal levels.

Another tested parameter is the number k of clusters to be generated. A decently accurate identification of all 5 considered traffic classes obviously requires at least 5 clusters, which is the default value considered. However, it results that just 3 clusters are generally sufficient to achieve good attack detection accuracy. If even higher numbers of clusters are used, all accuracy measures slightly increase, although the number of k-means iterations run by nodes and therefore the global complexity also does. About this aspect, apart from the number of clusters, conditions guaranteeing good accuracy (enough known classes per node and good network connectivity) often also entail a slightly lower average number of iterations per node.

Due to the lack of other works proposing decentralized data clustering for network security, a comparison of the results with the related literature is not fully feasible. However, the accuracy levels reached by experiments with distributed clustering are generally better or comparable to those obtained by other works performing standard centralized analysis, either supervised or not. The proposed system has the advantage over supervised ones of being able to distinguish previously unseen types of traffic and, given the distributed nature, it can be run on a network of nodes causing minimal traffic among them.

Bibliography

[1] Charu C Aggarwal and ChengXiang Zhai. A survey of text clustering algorithms. In *Mining Text Data*, pages 77–128. Springer, 2012.

[2] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27. Association for Computational Linguistics, 2009.

[3] Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining. In *LREC*, volume 10, pages 2200–2204, 2010.

[4] Paul N. Bennett and Nam Nguyen. Refined experts: improving classification in large taxonomies. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 11–18, 2009.

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent Dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[6] John Blitzer, Ryan McDonald, and Fernando Pereira. Domain adaptation with structural correspondence learning. In *Proceedings of the*

2006 conference on empirical methods in natural language processing, pages 120–128. Association for Computational Linguistics, 2006.

[7] Stephan Bloehdorn and Andreas Hotho. Boosting for text classification with semantic features. *Lecture Notes in Computer Science*, 3932:149, 2006.

[8] 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.

[9] Harold Borko and Myrna Bernick. Automatic document classification. *J. ACM*, 10(2):151–162, April 1963.

[10] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152, New York, NY, USA, 1992. ACM.

[11] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[12] Alexander Budanitsky and Graeme Hirst. Evaluating wordnet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

[13] J. B D Cabrera, L. Lewis, Xinzhou Qin, Wenke Lee, R.K. Prasanth, B. Ravichandran, and R.K. Mehra. Proactive detection of distributed denial of service attacks using MIB traffic variables-a feasibility study. In *Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on*, pages 609–622, 2001.

[14] Lijuan Cai and Thomas Hofmann. Text categorization by boosting automatically extracted concepts. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 182–189, 2003.

[15] Lijuan Cai and Thomas Hofmann. Hierarchical document categorization with support vector machines. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management*, pages 78–87, 2004.

[16] William B. Cavnar and John M. Trenkle. N-gram-based text categorization. In *In Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[17] Michelangelo Ceci and Donato Malerba. Classifying web documents in a hierarchy of categories: a comprehensive study. *Journal of Intelligent Information Systems*, 28(1):37–78, 2007.

[18] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Hierarchical classification: combining Bayes with SVM. In *Proceedings of the 23rd international conference on Machine learning*, pages 177–184, 2006.

[19] Nicolò Cesa-Bianchi, Claudio Gentile, and Luca Zaniboni. Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research*, 7:31–54, 2006.

[20] S. Cheeti, A. Stanescu, and D. Caragea. Cross-domain sentiment classification using an adapted nive bayes approach and features derived from syntax trees. In *Proceedings of KDIR 2013, 5th International Conference on Knowledge Discovery and Information Retrieval*, pages 169–176, 2013.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[22] V.M. Crestana and N. Soparkar. Mining decentralized data repositories. Technical Report CSE-TR-385-99, University of Michigan, 1999.

[23] Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Sibun. A practical part-of-speech tagger. In *Proceedings of the third conference on Applied natural language processing*, pages 133–140. Association for Computational Linguistics, 1992.

[24] 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.

[25] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Transferring naive bayes classifiers for text classification. In *Proceedings of the AAAI '07, 22nd national conference on Artificial intelligence*, pages 540–545, 2007.

[26] Dipanjan Das and André FT Martins. A survey on automatic text summarization. *Literature Survey for the Language and Statistics II course at CMU*, 4:192–195, 2007.

[27] S. Datta, C.R. Giannella, and H. Kargupta. Approximate distributed k-means clustering over a peer-to-peer network. *IEEE Transactions on Knowledge and Data Engineering*, pages 1372–1388, 2008.

[28] Hal Daumé III. Frustratingly easy domain adaptation. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 256–263, 2007.

[29] Franca Debole and Fabrizio Sebastiani. Supervised term weighting for automated text categorization. In *Proceedings of the 18th ACM Symposium on Applied Computing*, pages 784–788, 2003.

[30] Scott C. Deerwester, Susan T Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[31] D.E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 13(2):222–232, 1987.

[32] Harris Drucker, S Wu, and Vladimir N Vapnik. Support vector machines for spam categorization. *Neural Networks, IEEE Transactions on*, 10(5):1048–1054, 1999.

[33] Susan Dumais and Hao Chen. Hierarchical classification of web content. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 256–263, 2000.

[34] Susan Dumais, John Platt, David Heckerman, and Mehran Sahami. Inductive learning algorithms and representations for text categorization. In *Proceedings of CIKM '98, 7th International Conference on Information and Knowledge Management*, pages 148–155. ACM, 1998.

[35] BJ Field. Towards automatic indexing: automatic assignment of controlled-language indexing and classification from free indexing. *Journal of Documentation*, 31(4):246–265, 1975.

[36] Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM, 2001.

[37] Mark Alan Finlayson. Java libraries for accessing the princeton wordnet: Comparison and evaluation. In *Proceedings of the 7th Global Wordnet Conference, Tartu, Estonia*, 2014.

[38] George Forman. An extensive empirical study of feature selection metrics for text classification. *Journal of Machine Learning Research*, 3:1289–1305, 2003.

[39] William J Frawley, Gregory Piatetsky-Shapiro, and Christopher J Matheus. Knowledge discovery in databases: An overview. *AI magazine*, 13(3):57, 1992.

[40] Evgeniy Gabrilovich and Shaul Markovitch. Feature generation for text categorization using world knowledge. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1048–1053, 2005.

[41] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. In *IJCAI*, volume 7, pages 1606–1611, 2007.

[42] Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 283–291. ACM, 2008.

[43] Vishal Gupta and Gurpreet Singh Lehal. A survey of text summarization extractive techniques. *Journal of Emerging Technologies in Web Intelligence*, 2(3):258–268, 2010.

[44] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[45] Mark A Hall. *Correlation-based feature selection for machine learning.* PhD thesis, The University of Waikato, 1999.

[46] Eui-Hong Sam Han, George Karypis, and Vipin Kumar. *Text categorization using weight adjusted k-nearest neighbor classification.* Springer, 2001.

[47] Donna Harman. Overview of the first trec conference. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 36–47. ACM, 1993.

[48] Graeme Hirst and David St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. *WordNet: An electronic lexical database*, 305:305–332, 1998.

[49] Thomas Hofmann. Probabilistic latent semantic analysis. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 289–296. Morgan Kaufmann Publishers Inc., 1999.

[50] Jiayuan Huang, Alexander J Smola, Arthur Gretton, Karsten M Borgwardt, and Bernhard Schölkopf. Correcting sample selection bias by unlabeled data. *Advances in neural information processing systems*, 19:601–608, 2007.

[51] Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proc of 10th International Conference on Research in Computational Linguistics, ROCLING97.* Citeseer, 1997.

[52] Thorsten Joachims. A probabilistic analysis of the Rocchio algorithm with TFIDF for text categorization. In *Proceedings of ICML '97, 14th International Conference on Machine Learning*, pages 143–151, 1997.

[53] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. *Proceedings of ECML-98, 10th European Conference on Machine Learning*, 1398:137–142, 1998.

[54] P. Kabiri and A.A. Ghorbani. Research on intrusion detection and response: A survey. *International Journal of Network Security*, 1(2):84–102, 2005.

[55] Anne Kao and Steve R Poteet. *Natural language processing and text mining*. Springer, 2007.

[56] Athanasios Kehagias, Vassilios Petridis, Vassilis G Kaburlasos, and Pavlina Fragkou. A comparison of word-and sense-based text categorization using several classification algorithms. *Journal of Intelligent Information Systems*, 21(3):227–247, 2003.

[57] Ashraf M Kibriya, Eibe Frank, Bernhard Pfahringer, and Geoffrey Holmes. Multinomial naive bayes for text categorization revisited. In *AI 2004: Advances in Artificial Intelligence*, pages 488–499. Springer, 2005.

[58] Svetlana Kiritchenko and Stan Matwin. Email classification with co-training. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, pages 301–312. IBM Corp., 2011.

[59] Svetlana Kiritchenko, Stan Matwin, and A. Fazel Famili. Functional annotation of genes using hierarchical text categorization. In *in Proc. of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology (held at ISMB-05*, 2005.

[60] Man Lan, Chew Lim Tan, and Hwee-Boon Low. Proposing a new term weighting scheme for text categorization. In *AAAI*, volume 6, pages 763–768, 2006.

[61] S. le Cessie and J.C. van Houwelingen. Ridge estimators in logistic regression. *Applied Statistics*, 41(1):191–201, 1992.

[62] Wenke Lee and Salvatore J Stolfo. Data mining approaches for intrusion detection. In *7th USENIX Security Symposium*, 1998.

[63] Yoong Keok Lee and Hwee Tou Ng. An empirical evaluation of knowledge sources and learning algorithms for word sense disambiguation. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 41–48. Association for Computational Linguistics, 2002.

[64] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In

Proceedings of the 5th Annual International Conference on Systems Documentation, SIGDOC '86, pages 24–26, New York, NY, USA, 1986. ACM.

[65] David D Lewis. An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–50. ACM, 1992.

[66] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Machine learning: ECML-98*, pages 4–15. Springer, 1998.

[67] 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.

[68] David D Lewis, Yiming Yang, Tony G Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *The Journal of Machine Learning Research*, 5:361–397, 2004.

[69] Lianghao Li, Xiaoming Jin, and Mingsheng Long. Topic correlation analysis for cross-domain text classification. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[70] Wei Li and Andrew McCallum. Pachinko allocation: Dag-structured mixture models of topic correlations. In *Proceedings of the 23rd international conference on Machine learning*, pages 577–584. ACM, 2006.

[71] Hung-Jen Liao, Chun-Hung Richard Lin, Ying-Chih Lin, and Kuang-Yuan Tung. Intrusion detection system: A comprehensive review. *Journal of Network and Computer Applications*, 36(1):16–24, 2013.

[72] Xiao Ling, Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. Spectral domain-transfer learning. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 488–496. ACM, 2008.

[73] Xiao Ling, Gui-Rong Xue, Wenyuan Dai, Yun Jiang, Qiang Yang, and Yong Yu. Can chinese web pages be classified with english data source? In *Proceedings of the 17th international conference on World Wide Web*, pages 969–978. ACM, 2008.

[74] Bing Liu. Sentiment analysis and opinion mining. *Synthesis Lectures on Human Language Technologies*, 5(1):1–167, 2012.

[75] Tie-Yan Liu, Yiming Yang, Hao Wan, Hua-Jun Zeng, Zheng Chen, and Wei-Ying Ma. Support vector machines classification with a very large-scale taxonomy. *SIGKDD Explorations Newsletter*, 7(1):36–43, 2005.

[76] Yuanchao Liu, Xiaolong Wang, Zhiming Xu, and Yi Guan. A survey of document clustering. *Journal of Chinese Information Processing*, 20(3):55–62, 2006.

[77] Annie Louis and Ani Nenkova. Automatically assessing machine summary content without a gold standard. *Computational Linguistics*, 39(2):267–300, 2013.

[78] Hans Peter Luhn. A business intelligence system. *IBM Journal of Research and Development*, 2(4):314–319, 1958.

[79] Qiming Luo, Enhong Chen, and Hui Xiong. A semantic term weighting scheme for text categorization. *Expert Systems with Applications*, 38(10):12708–12716, 2011.

[80] Melvin Earl Maron. Automatic indexing: an experimental inquiry. *Journal of the ACM (JACM)*, 8(3):404–417, 1961.

[81] George A Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, 1995.

[82] Thomas P Minka. A comparison of numerical optimizers for logistic regression. `http://research.microsoft.com/en-us/um/people/minka/papers/logreg/`, 2003.

[83] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

[84] Roberto Navigli. Word sense disambiguation: A survey. *ACM Computing Surveys (CSUR)*, 41(2):10, 2009.

[85] Alexander Pak and Patrick Paroubek. Twitter as a corpus for sentiment analysis and opinion mining. In *LREC*, 2010.

[86] 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.

[87] Sinno Jialin Pan, James T Kwok, and Qiang Yang. Transfer learning via dimensionality reduction. In *Proceedings of the AAAI '08, 23rd national conference on Artificial intelligence*, pages 677–682, 2008.

[88] 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.

[89] Sinno Jialin Pan, Ivor W Tsang, James T Kwok, and Qiang Yang. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks*, 22(2):199–210, 2011.

[90] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

[91] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Foundations and trends in information retrieval*, 2(1-2):1–135, 2008.

[92] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. Thumbs up?: sentiment classification using machine learning techniques. In *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 79–86. Association for Computational Linguistics, 2002.

[93] Fuchun Peng, Dale Schuurmans, and Shaojun Wang. Augmenting naive bayes classifiers with statistical language models. *Information Retrieval*, 7(3-4):317–345, 2004.

[94] Xiaogang Peng and Ben Choi. Document classifications based on word semantic hierarchies. In *Proceedings of the International Conference on Artificial Intelligence and Applications*, pages 362–367, 2005.

[95] Emanuele Pianta, Luisa Bentivogli, and Christian Girardi. Developing an aligned multilingual database. In *Proc. 1st Intl Conference on Global WordNet*, 2002.

[96] Mohammad Taher Pilehvar, David Jurgens, and Roberto Navigli. Align, disambiguate and walk: A unified approach for measuring semantic similarity. In *ACL (1)*, pages 1341–1351, 2013.

[97] Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[98] Leonid Portnoy, Eleazar Eskin, and Sal Stolfo. Intrusion detection with unlabeled data using clustering. In *Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001)*, pages 5–8, 2001.

[99] Peter Prettenhofer and Benno Stein. Cross-language text classification using structural correspondence learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1118–1127, 2010.

[100] Xipeng Qiu, Wenjun Gao, and Xuanjing Huang. Hierarchical multiclass text categorization with global margin maximization. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 165–168. Association for Computational Linguistics, 2009.

[101] Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, 1989.

[102] Jason D Rennie, Lawrence Shih, Jaime Teevan, David R Karger, et al. Tackling the poor assumptions of naive bayes text classifiers. In *ICML*, volume 3, pages 616–623. Washington DC), 2003.

[103] Philip Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *Proceedings of the 14th international joint conference on Artificial intelligence-Volume 1*, pages 448–453. Morgan Kaufmann Publishers Inc., 1995.

[104] Miguel E. Ruiz and Padmini Srinivasan. Hierarchical text categorization using neural networks. *Information Retrieval*, 5:87–118, 2002.

[105] G. Salton. *The SMART Retrieval System—Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1971.

[106] Sam Scott and Stan Matwin. Text classification using wordnet hypernyms. In *Use of WordNet in natural language processing systems: Proceedings of the conference*, pages 38–44, 1998.

[107] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[108] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[109] Jr 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.

[110] George Siolas and Florence d'Alché Buc. Support vector machines based on a semantic kernel for text categorization. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 5, pages 205–209. IEEE, 2000.

[111] Masashi Sugiyama, Shinichi Nakajima, Hisashi Kashima, Paul Von Buenau, and Motoaki Kawanabe. Direct importance estimation with model selection and its application to covariate shift adaptation. In *Advances in Neural Information Processing Systems 20*, volume 7, pages 1433–1440, 2007.

[112] Aixin Sun, Ee-Peng Lim, Wee-Keong Ng, and Jaideep Srivastava. Blocking reduction strategies in hierarchical text classification. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1305–1308, 2004.

[113] Songbo Tan. Neighbor-weighted k-nearest neighbor for unbalanced text corpus. *Expert Systems with Applications*, 28(4):667–671, 2005.

[114] Xiaohui Tao, Yuefeng Li, RaymondY.K. Lau, and Hua Wang. Unsupervised multi-label text classification using a world knowledge ontology. In *Advances in Knowledge Discovery and Data Mining*, pages 480–492. Springer, 2012.

[115] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *The Journal of Machine Learning Research*, 2:45–66, 2002.

[116] Celine Vens, Jan Struyf, Leander Schietgat, Sašo Džeroski, and Hendrik Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, 73(2):185–214, 2008.

[117] Piek Vossen. *A multilingual database with lexical semantic networks*. Springer, 1998.

[118] Pu Wang, Carlotta Domeniconi, and Jian Hu. Using Wikipedia for co-clustering based cross-domain text classification. In *ICDM '08, 8th IEEE International Conference on Data Mining*, pages 1085–1090. IEEE, 2008.

[119] Evan Wei Xiang, Bin Cao, Derek Hao Hu, and Qiang Yang. Bridging domains using world wide knowledge for transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(6):770–783, 2010.

[120] 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.

[121] Gui-Rong Xue, Dikan Xing, Qiang Yang, and Yong Yu. Deep classification in large-scale text hierarchies. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 619–626, 2008.

[122] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information retrieval*, 1(1-2):69–90, 1999.

[123] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM*

SIGIR conference on Research and development in information retrieval, pages 42–49. ACM, 1999.

[124] Yiming Yang and Jan O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning*, pages 412–420, 1997.

[125] Yiming Yang, Jian Zhang, and Bryan Kisiel. A scalability analysis of classifiers in text categorization. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 96–103. ACM, 2003.

[126] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics, 1995.

[127] Jaehak Yu, Hansung Lee, Myung-Sup Kim, and Daihee Park. Traffic flooding attack detection with SNMP MIB using SVM. *Computer Communications*, 31(17):4212–4219, 2008.

[128] Bianca Zadrozny. Learning and evaluating classifiers under sample selection bias. In *Proceedings of the 21st International Conference on Machine Learning*, page 114. ACM, 2004.

[129] Fuzhen Zhuang, Ping Luo, Hui Xiong, Qing He, Yuhong Xiong, and Zhongzhi Shi. Exploiting associations between word clusters and document classes for cross-domain text categorization. *Statistical Analysis and Data Mining*, 4(1):100–114, 2011.