Reducing Feature Space for Learning to Rank in Cross-Language Information Retrieval

by

Schigehiko Schamoni

A thesis submitted in partial satisfaction of the
requirements for the degree of
Master of Arts

in

Computational Linguistics

Ruprecht-Karls-Universität Heidelberg
Neuphilologische Fakultät

Supervisor:

Professor Dr. Stefan Riezler

Heidelberg, September 2013

Abstract

Reducing Feature Space for Learning to Rank in
Cross-Language Information Retrieval

by

Schigehiko Schamoni

In this thesis we will examine a recent approach in the field of *information retrieval*. Information retrieval as a research field denotes the development and implementation of methods and techniques to retrieve information from a collection. The ranking component, one of the most important modules in today's search engines, is our central concern and we will employ *learning to rank*, a machine learning technique for improving ranking quality with respect to a certain information need.

The technique we will examine is based on a quadratic expansion of *word features* which is able to capture natural language phenomena such as polysemy and synonymy. This quadratic expansion implicates a high demand for computer resources regarding computational complexity and memory requirements. A central question is how to reduce these requirements while preserving the properties of the model. One way to address these problems is to employ *hashing* to directly reduce the memory requirements of the model. We will further examine several methods that reduce the complexity of the data, i.e. the number of dimensions of the data representation.

We will first apply our model to a monolingual task on a standard dataset for text-based retrieval and evaluate various methods for *feature reduction* and *feature selection*. In this task, our model performs very well and surpasses the results of experiments from other works on the same dataset considerably.

In the second part of our experiments we will transfer the model into a *cross-language* setting where we train the system on relevance data between Japanese and English patents. These relevances were created based on information from a citation graph. Our model is able to learn a relevance metric and simultaneously learn useful word relations *across* languages without any external translation resource. We will then enrich the system in a novel way by integrating highly informative domain knowledge features and receive improvements over state-of-the-art statistical machine translation based approaches for patent retrieval on the same dataset.

Finally, we will demonstrate the flexibility of our approach by additionally incorporating a completely different model into our system. The final results of the combined system show a significant gain over the retrieval performance of each single system.

Zusammenfassung

In dieser Arbeit untersuche ich einen modernen Ansatz zur Implementierung eines zentralen Moduls aus dem Bereich *Information Retrieval* (Informationsrückgewinnung, Informationsakquise). Information Retrieval beschreibt das Themengebiet, das sich mit der Entwicklung von Algorithmen, Techniken und Methoden der Akquise von digitalen Informationen beschäftigt, wie sie etwa in heutigen Suchmaschinen eingesetzt werden.

In dem von mir untersuchten Modul werden *Machine Learning* Techniken (maschinelle Lernverfahren) benutzt, um die Ergebnisse von traditionellen Suchverfahren zu verbessern. Hierzu wird in einer *Ranking*-Komponente die Reihenfolge der Suchergebnisse mit mathematischen Optimierungsmethoden den Anforderungen der Applikation entsprechend angepasst. Dieser Ansatz heißt *Learning to Rank* und ist heutzutage eine Standardmethode zur Optimierung der Suchergebnisse von Suchmaschinen.

Im Zentrum meiner Arbeit steht eine moderne Variante eines solchen Ranking-Verfahrens, das nicht nur die Einzelwörter aus Suchanfragen und Suchdokumenten auswertet, sondern auf einer durch quadratische Expansion der Wörter angereicherten Datenrepräsentation arbeitet.

Diese quadratische Expansion hat interessante Eigenschaften und kann Phänomene erfassen, die besonders in menschlicher Sprache auftreten. Allerdings bringt der Ansatz auch Probleme bezüglich Speicherkapazität, Lernkomplexität und Kapazität des Modells mit sich. Eine wichtige Frage ist daher, wie man diese Anforderungen verringern kann, ohne dabei die Vorteile des Modells aufzugeben. Hierzu reduziere ich auf der einen Seite die Anzahl der Variablen des Modells durch *Hashing*, und untersuche auf der anderen Seite, wie die Anzahl der Dimensionen der Daten sinnvoll verringert werden kann.

Das Model wird zunächst mit einer einsprachigen Anwendung evaluiert und die Auswirkungen der Methoden untersucht. Daraufhin wende ich das Modell in einer sprachübergreifenden Anwendung an, in der für ein gegebenes japanisches Patent relevante amerikanische Patente gefunden werden müssen. Diese Daten wurden auf Basis von Zitationen in den Patenten erstellt. Ich zeige dabei, dass das Modell ohne die Integration eines Übersetzungssystems Wortbedeutungen über Sprachgrenzen hinweg erkennt und gleichzeitig die Relevanz solcher Wortpaare im Bezug auf Patente beurteilen kann. Die erzielten Ergebnisse sind vergleichbar mit anderen modernen Methoden, die auf Unigramm-Ansätzen basieren und auf den gleichen Daten evaluiert wurden.

Dieses Modell kombiniere ich in einer neuartigen Weise mit speziell modellierten *Features* (Merkmalen) aus der Patentdomäne. Die erzielten Ergebnisse dieses Modells übertreffen die Ergebnisse von Methoden, die auf statistischen maschinellen Übersetzungssystemen basieren.

Das Besondere an dem Ansatz ist seine Flexibilität. Ich zeige wie es ohne Weiteres möglich ist, neue Module zu integrieren und dadurch die Leistung des kombinierten Systems zu steigern. Da ich diese Kombination mit Machine Learning Techniken optimiere, liefert das neue System zum Teil erheblich bessere Ergebnisse als jedes beteiligte Einzelsystem allein.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Searching information has become an everyday task for us. With the growing amount of electronic media accessible for many people, information retrieval (IR) has become an important component in our daily life. We use information retrieval systems in the form of web search engines often many times a day, and there are numerous applications specialized on certain information needs, e.g. querying databases for flight-schedule information, stock management software, medical applications, etc. Long before the development of the Internet, the retrieval of electronic information was already an active research field.

Although all applications today work with similar basic components (e.g. the vector space model, query expansion, latent representations), the ever growing amount of data requires the systems to sort and categorize information according to the application. A step forward to improve a system's retrieval quality is the definition of a real valued score that assesses relevance of a document with respect to a query. The idea is that retrieval is not only about term-matching, but that terms also have different weights (as in TFIDF, see Section 2.1). Finding the documents that fit best to a query has become the task of searching similar documents by means of a similarity metric. The user is then presented a list of results, starting with the documents that the system ranks most important, i.e. assigns the highest relevance or similarity score to.

However, all TFIDF-like systems (e.g. BM25, Jones et al. 2000) are static in their calculation of relevance, which means they cannot change their behavior even if they repeatedly present the user a ranking he is not satisfied with. Such user feedback, called clickthrough-data in the field of web-search, is the most valuable resource for commercially run search systems (Joachims, 2002). The biggest step towards a dynamic search system was the implementation of a ranking module that analyzes such feedback and tunes relevance assessments that are repeatedly corrected by the users. The method of using machine learning techniques to implement this ranking module is called learning to rank (LTR).

We follow the categorization of Li (2011) and take his ranking taxonomy to locate our own approach. According to Figure 1.1, ranking can nowadays be divided into two main areas, *rank creation* and *rank aggregation*. Each of these areas can further be divided into supervised and unsupervised approaches. Our work covers both main areas, i.e. creation

and aggregation, but we employ machine learning and thus follow strictly the supervised approach.

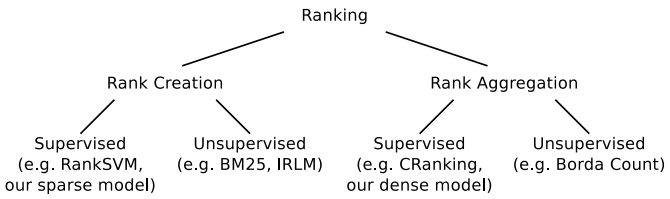

Figure 1.1: A taxonomy of different ranking systems. Idea motivated by: Li (2011).

Parallel to the development of *monolingual IR*, i.e. retrieval of information in a single language, methods that aim at IR across languages have been established: the task of searching information within a collection in a language different to the query language is called *cross-language information retrieval (CLIR)*. As the need for such applications is not obvious, we are providing some examples taken from Peters et al. (2012), which illustrate reasonable uses for CLIR:

- a researcher searching for all available information on a topic without limiting his search to certain languages
- a journalist searching for news articles and stories in a foreign language
- an investor acquiring first-hand knowledge about a region or country
- a traveler looking for nearby activities or local information
- a patent lawyer searching for relevant patents to register a customer's patent in a foreign country

With the growth of information electronically available in many different languages, the need for such applications is also growing.

One sign of the growing importance and the increasing interest in cross-language information applications is the success of the CLEF Initiative[1] (Conference and Labs of the Evaluation Forum, formerly known as Cross-Language Evaluation Forum). The CLEF Initiative defines tracks and provides datasets for multilingual retrieval, mostly in European languages. However, since their name changed in 2010 the tracks also have moved from a cross-language focus to a wider range of different information retrieval applications.

Another important collection of increasing popularity in recent years is the NII Test Collection for IR Systems (NTCIR). The corresponding Workshop[2] is an annual event that is usually held in Japan. The NTCIR defines cross-language tasks on information retrieval,

[1] `http://www.clef-campaign.org/`, accessed 10.09.2013
[2] `http://ntcir.nii.ac.jp/about/`, accessed 10.09.2013

question answering and text summarization focused on East Asian and European languages. We use parts of the NTCIR test collection on patent machine translation in this work.

A common way to address the CLIR problem is to extend an existing well-functioning IR system by translating certain components of the system. Peters et al. (2012) distinguish three different forms of translations that can be applied here:

- Query translation: the query is translated into the language of the document collection.
- Document translation: the document is translated into the language of the query.
- Both query and document translation: both languages are translated into a (not necessarily artificial) "pivot language".

Systems based on statistical machine translation (SMT) that follow the first approach are the ones most often built today. The second approach cannot be applied in most cases, because the computational cost of translating whole document collections is usually too high. The first approach, however, is examined in various works, see e.g. Jehl et al. (2012), Hieber et al. (2013) and Türe et al. (2012). The underlying strategy is often to take a standard retrieval algorithm and combine it cleverly with an SMT-system. Such approaches can benefit from integrated query expansion, where one term in the source language is replaced by multiple terms in the target language.

The method we present is more related to the third approach, although we do not model any underlying pivot language explicitly. Bai et al. (2010) present a model comparable to ours where a matrix, the central component of the model, is decomposed and the resulting two matrices project the query and document independently into a lower dimensional "pivot language"-space. This idea is taken up again and described in detail in Chapter 6 on future work.

Our contribution is to present a CLIR system which follows a different approach than most CLIR systems today. Instead of concentrating on the cross-language part and tackling the problem by applying statistical machine translation (SMT) in combination with standard IR methods, we employ a completely supervised approach that learns a relevance metric for ranking *across* languages without the need of parallel sentences. In this sense, our model is more directly focused on the CLIR problem as a whole and can optimize both word relations and document relevance simultaneously.

The final system we built combines sparse word-features and dense domain-specific features in a manner that has not been done before. The resulting model is *versatile* as it can be applied to any cross-language task where relevance matrices exist, and it is *extendable* as it can integrate any ranking system that generates scores on query-document pairs.

1.1 Motivation

The basic model we examine is closely related to the one presented by Bai et al. (2010), Chen et al. (2010), and Sokolov et al. (2013). This model is mathematically very clear

and has interesting properties regarding the ability to capture natural language phenomena like polysemy and synonymy. These phenomena can also be captured by lower dimensional representations such as *latent semantic indexing (LSI)* (Deerwester et al., 1990), however, LSI is an unsupervised technique and our approach works on a supervised signal to extract useful correlations.

The problem of the basic model we present in Section 2.3 is its high demand in terms of resource requirements. As the models works on a quadratic expansion of term-pairs, the memory requirements quickly become infeasible when applied to real-world datasets.

One way to address this problem is to employ a sparse representation of the model in memory. Chen et al. (2010) successfully show how ℓ_1-regularization (Section 2.2) helps to maintain a sparse model that has also improved generalization capabilities. We will examine ℓ_1-regularization in a similar task for comparison and in a completely different task on cross-language retrieval and evaluate in which cases this sparsity has the desired properties.

Another way to significantly reduce memory requirements is to apply *hashing*. The works by Shi et al. (2009) showed that approximation of complex functions by randomization is a technique widely used in many research areas and that hashing for machine learning perfectly fits into this scheme. We examine hash kernels as a method to considerably reduce the memory requirements of our model. The interesting question is whether we can implement hashing in a way that, instead of increasing information loss, it increases retrieval quality. We will further examine which other classic filtering methods are suited to reduce feature space without harming the quality of the final retrieval model.

A new idea proposed by Bai et al. (2010) is *correlated feature hashing (CFH)*, where hash kernels are applied to words using a semantically motivated strategy. We will examine under which conditions CFH is able to reduce feature space without losing important information contained in the data. Thus, a central question of our work is how to reduce feature space and at the same time improve retrieval quality.

Parallel to our examination of methods on a classic dataset for IR applications, we will build and evaluate a system aimed at searching patents. Patent retrieval is an economically highly relevant task. Patent infringement due to unsuccessful prior art search can endanger the existence of whole companies. Although the necessity of cross-language approaches was emphasized by Sarasúa and Corremans (2000) many years ago, a lot of work on patent retrieval is still done on monolingual data first (Guo and Gomes 2009, Bashir and Rauber 2010) and then extended to a cross-language application using (statistical) machine translation systems (Piroi et al., 2011).

We will evaluate the basic model in a cross-language information retrieval (CLIR) task on patents. A question is whether well-working methods from monolingual retrieval can be applied to the cross-language task. Our task setups for the monolingual IR on a standard dataset and for the cross-language patent domain are classic applications of supervised rank creation (Li, 2011).

Motivated by the work of Guo and Gomes (2009), we have created a model optimized for patent retrieval that combines both our sparse model and highly informative dense features specifically defined on the patent domain. This combination of sparse and dense features is

a novel approach as far as we know.

Finally, by extending the idea of Wu et al. (2008) on an enhanced form of supervised rank aggregation (see Figure 1.1), we examine the question whether our system would be flexible enough to incorporate other, probably completely different ranking approaches to further increase retrieval quality.

1.2 Structure of the Thesis

This thesis is divided into three main parts, of which the first explains theoretical methods and implementation issues, the second part describes monolingual experiments and is thus practically oriented, and the third part on cross-language experiments contains in addition to the practical part the explanation of a new idea and its implementation.

In the first part, we started in Chapter 2 with an explanation of the idea behind information retrieval and the components involved in standard systems, most notably the ranking component. We will then explain the theoretical background of machine learning and how this technique is applied in the "learning to rank" framework. We will then develop our basic model and present methods for feature reduction and feature selection because the memory and computational complexity is the main issue of our model. We conclude this part with a technical description of the architecture we use and the implementation in Chapter 3.

The second part, provided by Chapter 4, examines the practical applicability of our model in a monolingual retrieval task. After conducting preliminary experiments we will start evaluating the methods for feature reduction and feature selection proposed in the previous part. The expectation is to acquire experience with the model and to obtain results that can then be compared to the experiments in the next part of this work.

Chapter 5 is the third main part, where we evaluate the system in a cross-language information retrieval task on patent data. We will show how the model from the monolingual task can be successfully applied in a cross-language setting and evaluate several techniques for reducing model complexity. The development of a novel approach which combines the model with highly informative domain knowledge features is the most important section of this part. We will finish this chapter with a proof-of-concept experiment where we show that our final system is able to successfully integrate completely different systems.

Finally, in Chapter 6 we will discuss our findings and present possible extensions of our approach which we were considering during our work. In the last chapter of this thesis, Chapter 7, we will review the research questions that we brought up in the current chapter.

Chapter 2

Methods

This chapter provides an introduction to the theory and the methods examined in our work. We first explain in Section 2.1 the idea behind *information retrieval* and the components involved in standard systems with a focus on the ranking component. Section 2.2 introduces the methods needed for *learning to rank*, the application of machine learning to ranking. An important element in this framework are *word features*, and in Section 2.3 we describe how to extend the idea and develop our basic model. Due to the complexity of this model, it is necessary to approximate the problem using *hash kernels*, a method we explain in Section 2.4. Finally, Section 2.5 describes general strategies for *feature selection and reduction* and which dangers are involved in their application. In addition to this, we locate the methods we examine in our experiments in the feature selection and reduction context.

2.1 Information Retrieval

The motivation behind Information Retrieval (IR) is very old. IR in its basic sense exists since mankind has collected information to retrieve it at a later time. Looking up a recipe in a cookbook, reading the credit card number for payment, finding out on which weekday one's own birthday is next year – these are all applications of information retrieval. Information retrieval in its broadest sense is not bound to textual information. It can also mean to find out about the contents of a painting, the writing style of a text or the harmonies used in a musical work. To narrow the general idea of IR to the sense interesting for us, we will cite the definition given by Manning et al. (2008):

> "Information retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need from within large collections (usually stored on computers)."

Modern IR started in the 1950s (Singhal, 2001). When information became electronically available, techniques had to be developed to retrieve this information. With the rise of computers in the following decades the amount of information available for a single system

constantly grew, bounded only by the hardware resources available. Larger systems were already built by connecting computers together, usually for research purposes, which multiplied the available information within the network by the number of connected computers.

In the 1980s a new idea came up: nets of research computers should be connected among each other through a "meta-net". This idea was first realized as ARPANET and later reborn as the Internet (Roberts, 1986), which nowadays is – with all its pros and cons – an important motor for economic development and all kinds of research.

Parallel to this vast expansion of computer networks in size goes a natural increase of information available. This made the development of special techniques necessary which can handle this amount of information. These techniques to analyze, index, compress and retrieve information are all part of information retrieval.

Information retrieval in the area of computer science has been a text oriented domain. However, modern developments enable the application of retrieval algorithms on other types of data, e.g. pictures. For example, automatic labeling of persons in a photo collection, or object recognition apply algorithms very similar to those nowadays used in text-based information retrieval. Our work examines text-based IR, so when we talk about *information*, we usually mean the content of a document or a search query.

Indexing and Retrieval

To electronically retrieve information from a collection, the data needs to be available in an appropriate format. In the case of documents, the electronic representation is basically a large array of characters, a *string*, where words are separated by spaces, punctuation symbols, special symbols such as dashes, etc.

If we search for information (e.g. keywords) in a document like this thesis, we would use clever string matching algorithms like Knuth-Morris-Pratt (Knuth et al., 1977), but the same idea cannot be applied to thousands or millions of documents because searching would take too long. The difference is the scope of the search problem: in the case of a document, we want to find out the exact position of the search term, while in the case of a collection, we first need to know *which documents* are possible candidates before asking at *which position* the information is to be found.

Thus, for large-scale retrieval, i.e. retrieval of information within millions or billions of documents, we need a fast way to select possibly useful documents and a fast way to assess importance of the documents with respect to the information need expressed by the search query.

The issue of selecting possibly relevant documents in web-scale tasks is usually solved by querying an inverted index (Knuth, 1973). An inverted index is a simple but efficient data structure that can be applied to a variety of problems (see e.g. Section 3.2 on the construction of mappings for correlated feature hashing). It is basically a list of documents associated to a word. Figure 2.1 shows how an inverted index organizes terms and corresponding lists of document identifiers called *postings*.

Creating an inverted index is a straightforward problem: the algorithm first initializes an

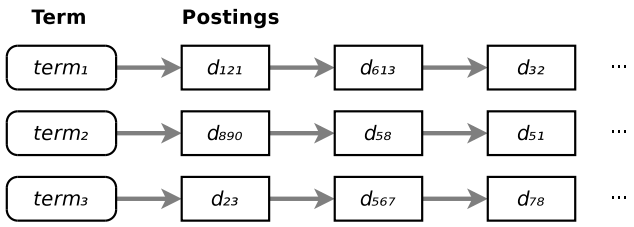

Figure 2.1: Structure of an inverted index with terms and associated lists of document IDs (postings).

array of words each with an empty list of postings. It then iterates over all documents and all words of the current document. For each word, it adds the document identifier (e.g. a pointer or a filename) to the list of postings for this particular word. After removal of duplicate entries in the postings list for each word the inverted index is ready. A comprehensive description of modern approaches together with a sophisticated implementation on a Hadoop-Cluster (see Section 3.1) that can be index collections of web-scale size is presented in Lin and Dyer (2010).

Retrieval of documents containing certain words is simple given an inverted index. For example, the information need is "*term1 term2*", then the system has to take the postings lists of *term1* and *term2*, build the intersection, and the set of remaining document identifiers is the result returned.

This type of searching is called *boolean retrieval*, because it only evaluates exact matches of words. It is the oldest form of computer based IR and nowadays only very few systems actively use boolean retrieval as an official search strategy (Manning et al., 2008, p. 14). A positive aspect is that it supports boolean expressions, so terms can be combined with logic operators such as *AND*, *NOT* and *OR*. In the example above, we assumed an implicit *AND* between the terms, which was implemented using a set intersection operation. *OR* can be implemented using the union, *NOT* using the complement. With a little modification, such systems can even support more sophisticated operators like *NEAR* for proximity searches.

However, plain boolean systems have several drawbacks. For example, they cannot distinguish between more and less important words, because in their logic every term has the same weight. What is even worse, they cannot see if a search word occurs one time or hundred times in a document, a number that will most certainly affect the relevance of the document. Thus, the boolean retrieval alone is usually too primitive to build a good system.

A more flexible model was described relatively early by Salton et al. (1975), but it took some years until computers had enough power to apply this method to retrieval tasks of larger scale. The basic idea behind the *vector space model* is that each document $\mathbf{d}_i$ is represented by vector of length $t = |\mathcal{D}|$, where $\mathcal{D}$ is the vocabulary of the collection:

$$\mathbf{d}_i = (t_{i,1}, \ldots, t_{i,t}) \tag{2.1}$$
$t_{i,j}$ representing the weight of the jth term

This treats a document as a unordered collection of words, a set, and projects it into a t-dimensional vector space. The dimensions are the weight of the term, which can be a binary measure, or, more sensible, the number of occurrences of the term. Two documents (i, j) can now be compared by computing a similarity coefficient $\text{sim}(\mathbf{d}_i, \mathbf{d}_j)$. This similarity measure can be arbitrarily complex, but in most cases a simple measure like cosine similarity returns very good results:

$$\cos\theta = \frac{\mathbf{d}_i \cdot \mathbf{d}_j}{||\mathbf{d}_i|| \cdot ||\mathbf{d}_j||}$$

This expression can be further simplified if we require normalization of the vectors to unit length, effectively removing the nominator:

$$\text{sim}(\mathbf{d}_i, \mathbf{d}_j) = \langle \mathbf{d}_i, \mathbf{d}_j \rangle = \sum_{k=1}^{|\mathcal{D}|} t_{i,k} \cdot t_{j,k} = \mathbf{d}_i^\intercal \mathbf{d}_j \tag{2.2}$$

Here, $\langle \cdot, \cdot \rangle$ denotes the vector dot-product and thus estimating similarity between $\mathbf{d}_i$ and $\mathbf{d}_j$ reduces to calculating the inner product. Besides being computationally efficient and elegant, this approach has many positive aspects. When similarity is calculated between a query and a document, it is necessary to have only at least one matching term to assess a score. And additionally, the weighting of terms can be further refined.

Such a refinement is TFIDF weighting, where term weights are not only values based on their frequency in a document but also discounted by their overall count among all documents. The motivation behind this scheme is that terms frequently occurring in documents, e.g. function words such as "a", "the" or "and", are irrelevant for the retrieval task (Jones, 1972). Whereas documents that contain words rare in the collection are very relevant with respect to this word. Thus, according to TFIDF weighting, function words get very low weights, and highly informative words receive high weights:

$$\begin{aligned} \text{tfidf}_{t,\mathbf{d}} &= \text{tf}_{t,\mathbf{d}} \cdot \text{idf}_t \\ \text{where idf}_t &= \log \frac{N}{\text{df}_t} \end{aligned} \tag{2.3}$$

In the formula above, $\text{tf}_{t,\mathbf{d}}$ is the term frequency of term t in document $\mathbf{d}$, N is the total number of documents, and df_t is the document frequency of term t, i.e. the number of

documents where t occurs. TFIDF itself consists of two components, the term frequency (TF), and the inverse document frequency (IDF). As TFIDF weighting only affects the values inside the vectors, calculating similarity via dot-product stays exactly the same as defined by formula 2.2.

Alternatively to boolean and vector space models exists a third variant, the *probabilistic model*. We do not go into the details, because their application is not relevant for our work. However, probabilistic models are a very important basic method for information retrieval and thus must be mentioned here.

The difference between a vector space model and a probabilistic model is not that great. Probabilistic models that are used today like Okapi BM25 receive vectors as input, but they calculate the similarity by a formula based on probability theory rather than dot-products in vector space (Jones et al., 2000). The model assumes that $\text{sim}(q, d)$ is represented by a conditional probability distribution $P(R|q, d)$ where the relevance R can be either 1 (relevant) or 0 (irrelevant).

Another model based on probability distributions are *language models for IR (LMIR)*. In the basic LMIR, $\text{sim}(q, d)$ is represented by a conditional distribution $P(q|M_d)$ that expresses the probability of a query q being generated by a language model M_d defined by some document d. An introduction on this topic can be found in Manning et al. (2008, ch. 12).

Ranking

The vector space model (as well as the probabilistic models) returns a score indicating similarity. If the weighting is constructed in an clever way like with TFIDF, this score indeed reflects some relevance of one document to another, or of a document to a search query. In modern information retrieval, this "relevance" notion has to be further refined, because relevance is not only a (more of less) elaborate method of matching terms. Relevance can change over time, relevance is dependent on the application and on the type of collection – in short, calculating a similarity that can be used as a reasonable relevance measure needs more than just a vector space model and a dot-product.

BM25 is one of those approaches that aim to make the similarity score to better reflect a relevance measure. However, BM25 still calculates term weights on a given corpus and keeps them once and for all (as long as the collection does not change). A very well working solution to estimate importance of web pages is PageRank (Page et al., 1999). PageRank scores can be used to reorder a list returned by e.g. a vector space model. This way the final model not only evaluates document similarity but also the importance of target documents.

Adding new modules is the key to make the retrieval process more flexible. Such a module that optimizes the final ordering of retrieval results is named *ranking component*. Figure 2.2 illustrates its position in the most simple way to visualize a retrieval pipeline: It starts by comparing a query to the contents of a document collection. This might happen on a subset of the collection, e.g. if an indexing is implemented. Then, the system generates similarity or relevance scores for the candidate documents to finally output the sorted list of documents (or document identifiers). This ranking component is often quite complex; it can consist of

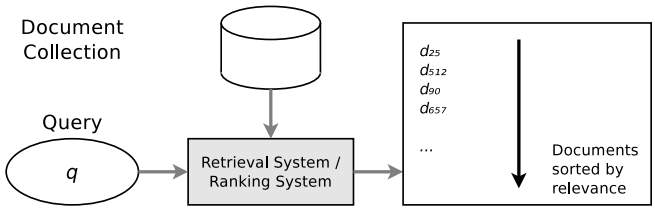

Figure 2.2: Function of the ranking component in a (very simplified) retrieval pipeline.

many submodules that are working together in a sophisticated way.

2.2 Learning to Rank

If several modules are combined, the next questions must be how to connect them. A natural choice to answer this question is to use machine learning techniques and let the system find the optimal combination by itself. This method has become a key technology in IR systems, because it can easily incorporate many modules or *features* and finds the optimal combination according to the intended purpose of the application.

For example in log files of web search engines, a lot of click-through data is collected, where the user actively gives feedback which documents suited his information need best. Evaluating such feedback for ranking of results is a very valuable source to create data for training an improved ranking system. Joachims (2002) describes how to build a ranking system based on a support vector machine (Cortes and Vapnik, 1995) that can efficiently learn a ranking function on partial user feedback.

Machine Learning

Learning to Rank (LTR) is a supervised task, which means we start with an input space $\mathcal{X}$ where the document or feature vectors exist and an output space $\mathcal{Y}$ containing labels or grades. The connection between an input element $\mathbf{x} \in \mathcal{X}$ and the corresponding label $y \in \mathcal{Y}$ is given by an unknown joint probability distribution $P(\mathcal{X}, \mathcal{Y})$. The task of LTR is to estimate or "learn" this probability distribution $P(\mathcal{X}, \mathcal{Y})$.

To asses a quality measure to the estimated probability distribution, the idea is to calculate the risk function $R(\cdot)$, which is the expected loss $l(\cdot)$ with respect to the true probability distribution $P(\mathcal{X}, \mathcal{Y})$ (Li, 2011):

$$R(f) = \int_{\mathcal{X}\times\mathcal{Y}} l(f(\mathbf{x}), y) \mathrm{d}P(\mathbf{x}, y) \tag{2.4}$$

Equation 2.4 applied to training data gives the empirical risk, which is the sum of losses l over the whole training set:

$$\hat{R}(f) = \frac{1}{m} \sum_{i=1}^{m} l(f(\mathbf{x}_i), y_i) \tag{2.5}$$

Finding the distribution $P(\mathcal{X}, \mathcal{Y})$ is now reduced to the problem of minimizing the empirical risk $\hat{R}(f)$. This phase of learning a function is called *training* and it is conducted on a certain partition of the whole data set, the *training set*.

After training, the ranking function is evaluated against a different partition of the data reserved for testing, the *test set*. The system basically calls the function $f(q, d)$ with the document data from the test partition and checks if the ranking function creates the expected ranking. This phase is called *test* or *testing*. In between these two phases are often several steps where optimal parameters have to be chosen empirically. In serious experiments, these optimizations must be conducted neither on training nor on test data but on a third partition, the *development set*. In some cases, e.g. in the experiments in Chapter 4, a development set is not available, in which case the training set is used for parameter estimation. Tuning settings on the test set is prohibitive, because it leads to unreliable results that do not apply to unseen data.

Optimization Problem

An optimization problem is a mathematically well-known problem and there usually exist many ways to find a solution. The selection of the appropriate solver depends on the type of problem, e.g. linear or quadratic, and on the properties of the function and the constraints, e.g. convex functions and convex sets (Boyd and Vandenberghe, 2004).

As Equation 2.5 is the key to turn the LTR task into an optimization problem, the question is what are the properties of the function and which are the constraints. For a start, the general optimization problem (OP) looks like this:

$$\min_{f} R(f) \tag{2.6}$$

This means, we are searching for a function f that minimizes the expected risk R. As we do not have access to all future data and furthermore this data is not labeled, we assume that our sample of labeled data is independently and identically distributed (i.i.d.) as the true distribution $P(\mathcal{X}, \mathcal{Y})$. We can now learn a function f based on the empirical risk and optimization problem 2.6 becomes $\min_f \frac{1}{m} \sum_{i=1}^{m} l(f(\mathbf{x}_i), y_i)$.

Assuming our data resides in an arbitrary complex feature space where we can find the expected ordering, learning to rank usually distinguishes three mutually exclusive approaches regarding the basic definition of the ranking problem (see e.g. Chapelle and Chang, 2011):

1. The *pointwise approach* treats the ranking problem as a classification, regression, or ordinal regression problem. The basic idea is to learn parallel hyperplanes that separate the differently ranked data. To estimate the rank of a given document, it is evaluated against each of the hyperplanes to find out to which union of halfspaces the

feature representation belongs. These partitions in feature space define the rank of the document in question. Although the pointwise approach can be implemented on practically all known linear models, it is computationally expensive.

2. The *listwise approach* is probably the most "natural" approach. Here, the idea is to directly work on lists of ranked documents for training and prediction. The loss function is thus defined over all documents corresponding to a query. The benefit of this approach is that the results are easier to analyze as the structure of such groups is maintained, and it can directly incorporate classic IR measures (see e.g. Section 3.3) into the loss function. The drawback is that the loss function quickly becomes very complex and, as IR measures are often discontinuous, the objective function sometimes cannot be optimized by gradient based techniques.

3. The third strategy is the *pairwise approach*. The idea behind this approach is to learn on the order of pairs. The training data for the learning system are pairs of documents and the goal is to learn a function f that reproduces the original ordering. This is the approach we follow in this work.

More information together with an extensive analysis of the point- and listwise approach can be found in Li (2011).

Pairwise Approach in Primal Space

One famous implementation of the pairwise approach is the RankSVM (ranking support vector machine) proposed by Joachims (2002). We start with the formulation of the unconstrained objective function for the RankSVM ($n = 2$) given by Sculley (2009):

$$\min_{\mathbf{w}} \frac{\lambda}{n}||\mathbf{w}||_n^n + \frac{1}{|\mathcal{T}|} \sum_{((\mathbf{q},\mathbf{d}_a,y_a),(\mathbf{q},\mathbf{b}_b,y_b))\in\mathcal{T}} (1 - \mathbf{w}^\intercal(\mathbf{d}_a - \mathbf{d}_b) \cdot \text{sign}(y_a - y_b))_+ \tag{2.7}$$

In this notation, $(c)_+ = \max(0, c)$ so the expression within the sum is actually the hinge-loss on a single example, $\mathbf{w}$ is the vector of parameters to learn, $\mathcal{T}$ is a set containing of all possible tuples of queries $\mathbf{q}$ and two documents $\mathbf{d}_a$ and $\mathbf{d}_b$ with their corresponding labels y_a and y_b (rank). The parameter λ is called the regularization parameter and controls how much weight should be given to the n-norm of the weight vector.

It is important to note that two documents $\mathbf{d}_a$ and $\mathbf{d}_b$ that belong to the same relevance group do not increase the sum as $\text{sign}(y_a - y_b) = 0$. Furthermore, $(\mathbf{d}_a - \mathbf{d}_b) \cdot \text{sign}(y_a - y_b) = (\mathbf{d}_b - \mathbf{d}_a) \cdot \text{sign}(y_b - y_a)$ which cuts the complexity of the OP by half.

The optimization problem 2.7 is a (convex) quadratic program (QP) which can be solved by a variety of algorithms. A few years ago, the standard approach to solve the QP was to transfer the primal problem into dual space and formulate the dual problem (see e.g. Schölkopf and Smola, 2002). The main reasons were that duality theory provides a convenient way to integrate the constraints into the dual problem, and that the dual problem

can be formulated with dot-products, making it possible to apply kernels (see Section 2.2). After solving the dual problem, the solution can be transferred back by applying the KKT-conditions (see e.g. Boyd and Vandenberghe 2004).

Chapelle (2007) showed that the QP of support vector machines (SVMs) (Cortes and Vapnik, 1995) can be efficiently trained in primal space and that in some cases solutions are even better because there is no theoretical guarantee that an approximate solution in dual space is a good approximate solution in primal space.

The formulation of the optimization problem 2.7 is already an unconstrained QP, and we further notice that if we restrict training to the half of pairs for which $f((\mathbf{q}, \mathbf{d}^+), y^+) > f((\mathbf{q}, \mathbf{d}^-), y^-)$, where $\mathbf{d}^+$ is a higher ranked document than $\mathbf{d}^-$, the sign$(\cdot)$ expression will always be 1 and can be left out. A final transformation is to treat the regularization expendable and what is left is basically a large margin perceptron as proposed by Collobert and Bengio (2004). For convenience, we define $\overline{\mathbf{d}}_j = (\mathbf{d}_j^a - \mathbf{d}_j^b)$ thus the loss for one training pair $(\mathbf{d}_j^a, \mathbf{d}_j^b)$ can be written as:

$$l_j(\mathbf{w}) = (1 - \mathbf{w}^\intercal \overline{\mathbf{d}}_j)_+ \tag{2.8}$$

Bottou and Bousquet (2008) report that in large-scale settings the strategy of stochastic gradient descent (SGD) provides efficient training and very good generalization performance. Sculley (2009) gives a clear formulation of the algorithm for the SGD approach of which we give a slightly simplified variant in Algorithm 1.

Algorithm 1 Stochastic Pairwise Descent.

1: $\mathbf{w}_0 \leftarrow \emptyset$
2: **for** $i = 1$ to t **do**
3: $\quad((\mathbf{q}_j, \mathbf{d}_j^+), (\mathbf{q}_j, \mathbf{d}_j^-)) \leftarrow$ GetRandomPair(D)
4: $\quad\overline{\mathbf{d}}_j \leftarrow (\mathbf{d}_j^- - \mathbf{d}_j^+)$
5: $\quad\mathbf{w}_i \leftarrow \mathbf{w}_{i-1} + \eta \nabla l_j(\mathbf{w}_i)$
6: **end for**
7: **return** $\mathbf{w}_t$

The function GetRandomPair(D) always returns a more ($\mathbf{d}_j^+$) and a lesser relevant document ($\mathbf{d}_j^-$) with respect to a query ($\mathbf{q}_j$). The sampling of documents is done according to a multipartite pairwise scheme, where pairs are randomly sampled from different relevance levels as shown in Figure 2.3: pairs are only sampled from $r_2 \circ r_0$, $r_2 \circ r_1$, or $r_1 \circ r_0$.

A further improvement related to magnitude-preserving ranking was suggested by Cortes et al. (2007). They define the *hinge rank loss* as a natural extension of the pairwise misranking loss in Equation 2.8. The central idea is to preserve the margin between documents that belong to rank levels that are not adjacent, e.g. $r_2 \circ r_0$ in Figure 2.3. For the pairwise misranking loss, this results in:

$$l_j(\mathbf{w}) = (b_j - \mathbf{w}^\intercal \overline{\mathbf{x}}_j)_+ \tag{2.9}$$

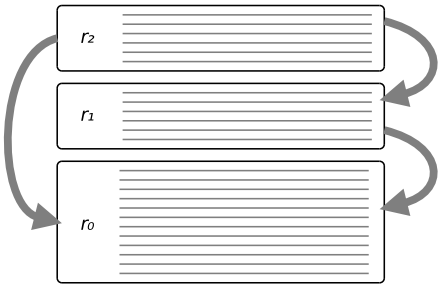

Figure 2.3: Multipartite pairwise ranking with three ranking levels r_2,r_1 and r_0.

Now, the final stochastic subgradient can be easily derived:

$$\nabla l_j(\mathbf{w}) = \begin{cases} -\overline{\mathbf{x}}_i & \text{if } \mathbf{w}^\intercal \mathbf{x}_i < b_i \\ 0 & \text{otherwise} \end{cases}$$

This step completes the list of required components to build the LTR system.

The training and test procedure of the final ranking system are depicted in Figure 2.4. In this figure, training instances consisting of queries q, pairs of documents (d^+, d^-) and pairs of labels (y^+, y^-), are sent to the learning system. The learning system estimates a function $f(q, d)$ based on the statistics of the data. This function, the *model*, is then used by the ranking system to evaluate test data, i.e. calculate rankings on previously unseen query-document pairs (q, d). The ordering given by these predictions is then compared to the expected ordering to estimate the quality of the model.

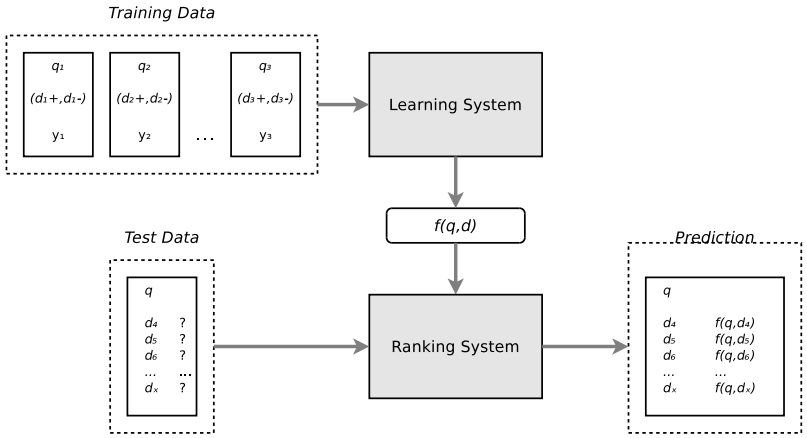

Figure 2.4: Pairwise learning to rank approach. Idea adopted from: Liu (2010)

Although SGD looks like a relatively poor optimization technique, Bottou and Bousquet (2008) pointed out that the generalization ability of SGD does not depend on the

training set size but on the number of gradient steps taken. Sculley (2009) examined the performance of the pairwise sampling approach and showed that efficient SGD learners reduce training time by orders-of-magnitude with no observable loss in performance. This makes stochastic gradient descent an excellent training strategy for large-scale learning to rank.

Regularization

At one point, we have simply discarded the regularization term present in the optimization problem 2.7 to show the connection between the ranking perceptron we use and the classical RankSVM. However, regularization is an important technique to tackle the problem of overfitting. The idea behind regularization is to influence the parameters of the function (i.e. weights of the vector) in a way that the learned model will not adapt too much to the training data. Two regularization schemes are frequently used, ℓ_1- and ℓ_2-regularization.

For the optimization problem 2.7, applying ℓ_2-regularization (also known as *ridge regression*) means setting $n = 2$ and using the 2-norm in the first term. This effectively pushes down larger weights and let smaller weights of $\mathbf{w}$ grow during optimization, resulting in a *dense model*, i.e. a model with few zero parameters.

The other frequently used scheme is ℓ_1-regularization, which was initially proposed as *Least Absolute Selection and Shrinkage Operator (LASSO)* by Tibshirani (1996). The parameter in the optimization problem 2.7 is $n = 1$, i.e. using the 1-norm in the first term. This means to push all weights towards 0 regardless of their value during optimization. The strategy results in a *sparse model*, a model with only few non-zero parameters. If we apply this idea to the optimization problem (2.7), the resulting objective function becomes:

$$\arg\min_{\mathbf{w}} \frac{1}{|\mathcal{T}|} \sum_{(q,d^+,d^-)\in\mathcal{T}} l_{\mathbf{w}}(q, d^+, d^-) + \lambda||\mathbf{w}||_1 \tag{2.10}$$

Any type of regularization must be carefully adjusted to the underlying task. It can support the learning task as in Section 5.2, or it can spoil it as in Section 4.2. In our setup, we evaluated ℓ_1-regularization to enforce sparsity of the model. Applying this regularization in the SGD framework is surprisingly simple. The update step in line 5 of Algorithm 1 with ℓ_1-regularization becomes:

$$\mathbf{w}_{i+1} = \mathbf{w} + \eta \frac{\partial}{\partial \mathbf{w}} (l_j(\mathbf{w}) - \frac{\lambda}{N} \sum_i |w_i|)$$

The only difficulty of getting a non-differentiable weight in the case $w_i = 0$ can be solved by setting a subgradient. However, there are many ways to implement ℓ_1-regularization in a computationally more efficient way. Here, we only demonstrate how the gradient step works in principle, but for a more detailed discussion, see e.g. Tsuruoka et al. (2009).

One final note on the regularization parameter λ: finding the right regularizer is expensive because the space of possible λ-settings has to be searched extensively, usually in log space.

However, approaches for automatic regularization exist, e.g. Obozinski et al. (2010), but implementing these ideas had to be moved to future work.

Kernels

In the machine learning framework, the basic assumption is that data contains some hidden pattern and certain methods are able to extract these patterns. Data is represented by features, or in other words, data is projected into a feature space, as with documents that are projected into a vector space.

The idea of having a feature representation that is the projection of some data into a feature space is very useful. This abstraction enables the use of alternative representations of data, for example when data is mapped into a not so apparent, probably very high dimensional feature space. A feature map from input space $\mathcal{X}$ into the feature space $\mathcal{H}$ is be defined by:

$$\begin{aligned} \Phi \ &: \mathcal{X} \to \mathcal{H} \\ &x \mapsto \mathbf{x} := \Phi(x) \end{aligned}$$

Having such a feature map, the kernel is the dot product in feature space:

$$\begin{aligned} k \ &: \mathcal{H} \times \mathcal{H} \to \mathbb{R} \\ &k(x, x') = \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}} \end{aligned} \tag{2.11}$$

This is basically the idea: Kernels calculate dot-products in a feature space. The exceptional property of kernels is that although they work in a feature space, the mapping into this space is hidden behind the kernel function and these can be applied to any algorithm that is based on dot-products, e.g. linear classifiers such as perceptron (Rosenblatt, 1958) or SVM, but also kNN-classifiers. Exchanging the dot-product by a kernel function and have the classifier work in a different feature space is called the *kernel trick*.

To complete the kernel definition in 2.11, $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ must respect the following properties to be an inner product:

1. Bilinear form: $\langle \alpha_1 f_1 + \alpha_2 f_2, g \rangle_{\mathcal{H}} = \alpha_1 \langle f_1, g \rangle_{\mathcal{H}} + \alpha_2 \langle f_2, g \rangle_{\mathcal{H}}$
2. Symmetry: $\langle f, g \rangle_{\mathcal{H}} = \langle g, f \rangle_{\mathcal{H}}$
3. Positive definiteness: $\langle f, f \rangle_{\mathcal{H}} \geq 0$, and $\langle f, f \rangle_{\mathcal{H}} = 0$ iff $f = 0$

Based on the inner product, we can now define a norm $||f||_{\mathcal{H}} = \sqrt{\langle f, f \rangle_{\mathcal{H}}}$. Spaces in which an inner product and a norm exist are called Hilbert spaces[1].

[1] We are simplifying the definition of Hilbert spaces here. For a precise definition, see e.g. Hunter and Nachtergaele (2001).

The reproducing kernel map is a special mapping that assigns to a point $x \in \mathcal{X}$ a kernel function $k(\cdot, x)$:

$$\begin{aligned} \Phi \ :&\mathcal{X} \to \mathbb{R}^{\mathcal{X}} \\ &x \mapsto k(\cdot, x) \end{aligned}$$

In other words, with $k(\cdot, x)$ having the dimensionality of $|X|$, each pattern is represented by its similarity to all other points in $\mathcal{X}$.

The idea is now to create a feature space associated with Φ as shown in Schölkopf and Smola (2002). The first step is to turn images of the input patterns under Φ into a vector space. This is accomplished by creating the linear combination

$$f(\cdot) := \sum_{i=1}^{m} \alpha_i k(\cdot, x_i)$$

with $m \in \mathbb{N}$, $\alpha_i \in \mathbb{R}$ and data points $x_1, \ldots, x_m \in \mathcal{X}$. The next step is to define a inner product $\langle f, g \rangle_{\mathcal{H}} := \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \beta_j k(x_i, x'_j)$, where additionally $m' \in \mathbb{N}$, $\beta_i \in \mathbb{R}$ and $x_1, \ldots, x_{m'} \in \mathcal{X}$, and show that it respects the properties of inner products listed above.

Then, based on these definitions, Schölkopf and Smola (2002) prove that the constructed kernel has the reproducing kernel property and thus k satisfies:

$$\langle f, k(\cdot, x) \rangle_{\mathcal{H}} = f(x) \tag{2.12}$$

Equation 2.12 is called the *reproducing property* and k is here the *representer of evaluation*. In particular, for any $x, x' \in \mathcal{X}$

$$k(x, x') = \langle k(\cdot, x), k(\cdot, x') \rangle_{\mathcal{H}}$$

Thus, with k being a dot-product between feature maps, $\Phi(x) = k(\cdot, x)$ is indeed a valid feature map and a reproducing kernel is a valid kernel according to the original definition given in 2.11.

As these reproducing kernel Hilbert spaces (RKHS) are spaces of functions such that all evaluation functionals are continuous, they provide a powerful framework for the problem of learning from data.

2.3 Word Features

In the classic vector space model for information retrieval reviewed in Section 2.1 a dot-product is calculated on document vectors to determine the similarity between two documents. The result of this calculation is independent from the application, as there are no tunable parameters that can influence the result. The learning to rank approach adds a machine learning component that enables the system to adapt to certain information needs. For both systems, the inputs are vectors that contain word frequencies as dimensions.

Quadratic Word Features

The basic LTR approach presented in Section 2.2 can be seen as learning a re-weighting of frequencies if it is applied to word vectors. These models, from TFIDF to the basic LTR approach based on word vectors, only work on matching words because non-matching words cannot contribute to the similarity score. Another implication of these bag-of-words models is that words are generally considered independent, an assumption that firstly downsizes the problem and secondly works surprisingly well in many NLP tasks. Especially with limited resources, this form of simplification was absolutely necessary until a few years.

Although successful, this independence assumption is definitely wrong in the context of word meanings. Words can have similar or opposite meanings, and often the context disambiguates the meaning of another word. As computers nowadays have access to constantly growing resources, it became possible to take more and more context information into account. One idea is to move from single word features to word pair features.

Consider a corpus of l documents and a vocabulary $\mathcal{D}$, the expression $\{\mathbf{d}_t\}_{t=1}^{l} \subset \mathbb{R}^{\mathcal{D}}$ denotes the set of documents and $q \in \mathbb{R}^{\mathcal{D}}$ is a query. In analogy to the model described by Equation 2.1, the jth component of $\mathbf{q}$ or $\mathbf{d}$ indicates the frequency of the word j, which can be raw counts or TFIDF values, and the whole vector is usually normalized to unit length. Using these definitions, we will now describe our basic model.

Basic Model

The basic model we consider is similar to the one initially proposed by Bai et al. (2010):

$$f(\mathbf{q}, \mathbf{d}) = \mathbf{q}^\intercal \mathbf{W} \mathbf{d} = \sum_{i,j=1}^{\mathcal{D}} q_i W_{ij} d_j \tag{2.13}$$

Here, the score $f(\mathbf{q}, \mathbf{d})$ expresses the relevance of a document $\mathbf{d}$ to a query $\mathbf{q}$.

Similarly to the construction of feature maps for kernels in Section 2.2 which transform data from the sample space into the feature space, the following joint feature map projects query-document pairs into a joint feature space:

$$\Phi_{((i-1)\mathcal{D}+j)}(\mathbf{q}, \mathbf{d}) = (\mathbf{q} \otimes \mathbf{d})_{ij} = (\mathbf{q}\mathbf{d}^\intercal)_{ij} \tag{2.14}$$

With $s = ((i-1)\mathcal{D} + j)$ being the dimension correlated to a certain word pair in the feature space, and $\mathbf{q}\mathbf{d}^\intercal$ being the outer product of the two vectors, the basic model can be rewritten as:

$$f(\mathbf{q}, \mathbf{d}) = \mathbf{w} \cdot \Phi(\mathbf{q}, \mathbf{d}) \tag{2.15}$$

This way, the matrix-vector and vector-vector operation simplifies to a standard dot-product and all methods and techniques described in Section 2.2 can be applied to this simplified formulation of the basic model.

Properties

The basic model has some interesting properties: it can capture polysemy and synonymy, and it captures dependencies between words, e.g. a word pair with high similarity can be discounted with a negative entry if one of its words happens to occur with another word that disambiguates the meaning. Assuming the documents are not too long, the model presented can efficiently learn such correlations from data.

However, there are problems regarding computational cost. Although Moore's law is still active (with some minor modifications of its original form) and the number of transistors is doubled every 18 month, the amount of memory required for this model is infeasible at the moment. Even for small corpora like the 20newsgroup corpus (see Section 4.1) the vocabulary size, i.e. the number of different types, reaches $\mathcal{D} = 125,000$. The model we propose uses a matrix of size $\mathcal{D} \times \mathcal{D}$ which results in a weight vector of size 62.5 GB assuming single precision, and twice the size for double precision. Today's computers can handle this, but on more complex corpora such as Wikipedia[2], the number of different tokens quickly exceeds millions and the space requirements grow up to multiple terabytes. This data has to be kept in random access memory, otherwise learning and evaluating the model will be slowed down by multiple orders of magnitude.

The high demands for resources and computational power require special solutions. As the query and document vectors are usually sparse, they can be represented in a compressed way in memory. However, the weight vector w cannot be easily compressed in the same way because during learning the vector is constantly changing and any compression technique will add a large overhead to the process.

During evaluation, computations can be reused as the system needs to calculate the expression $\mathbf{q}^\intercal \mathbf{W} \mathbf{d}$ for each candidate document $\mathbf{d}$, so it is advisable to calculate repeating terms only once. Bai et al. (2010) propose calculating the vector $\mathbf{v} = \mathbf{q}^\intercal \mathbf{W}$ once for each query, and thus $f(\mathbf{q}, \mathbf{d}) = \mathbf{v}^\intercal \mathbf{d}$ simplifies to a simple dot-product of vectors of dimension $\mathcal{D}$.

2.4 Hash Kernels

Kernels are a very powerful mechanism but there are issues. One problem is that using complex kernels, e.g. by combining multiple kernels to a new kernel, will eventually make any data linearly separable, but in the worst case this leads to overfitting because for example in an SVM, every data point might become a support-vector (Schölkopf and Smola, 2002). Another problem for using large expansions of kernels is runtime performance.

The definition of a kernel has been given in Section 2.2. In the following, we denote the domain of observations $\mathcal{X}$ and assume that $\mathcal{H}$ is a Reproducing Kernel Hilbert Space with kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$.

[2] `http://www.wikipedia.org/`, accessed 24.09.2013

Randomized Approximation

To address the two problems mentioned above in the introduction of this section, one line of research is to keep the kernel simple. According to Shi et al. (2009), the idea of approximating an expansion to get more desirable computational properties has been independently proposed by Kontorovich (2007) and Rahimi and Recht (2007). Let $c \in \mathcal{C}$ be a random variable with measure P, and let $\phi_c : \mathcal{X} \to \mathbb{R}$ be functions indexed by c. Then, a kernel of the form

$$k(x, x') = \mathbb{E}_{c \sim P(c)}[\phi_c(x)\phi_c(x')] \tag{2.16}$$

can be approximated by sampling a set $C = \{c_1, \ldots, c_n\} \sim \mathrm{P}$ and expanding

$$\overline{k}(x, x') = \frac{1}{n} \sum_{i=1}^{n} \phi_c(x)\phi_c(x')$$

This is equivalent to approximating the feature map $\phi_c(x)$ by $\overline{\phi}(x) = n^{-\frac{1}{2}}(\phi_{c_1}(x), \ldots, \phi_{c_n}(x))$. It is further possible to get Chernoff bounds for large deviations between $\overline{k}(x, x')$ and $k(x, x')$ if $\phi_c(x)\phi_c(x')$ is bounded, i.e. $\phi_c(x)\phi_c(x') \in [a, a + r]$ for all c, x and x'. Shi et al. (2009) point out that this strategy has widespread applications, from which we name only three:

- Kontorovich (2007) shows how to design kernels on regular languages by sampling from the class of languages.
- Schölkopf and Smola (2002) construct an empirical kernel map with $\mathcal{C} = \mathcal{X}$, $\mathrm{P}(c) = \mathrm{P}(x)$, and utilizes a kernel function κ with the property $\phi_c(x) = \kappa(c, x)$. In other words, the sample points c_i are placed on the training data.
- An early instantiation of Equation 2.16 was proposed by Watkins (2000) for the pair HMM kernel $k(x, y) = P(x, y|c)$. In this case, $\mathcal{X}$ and $\mathcal{C}$ are domains of biological sequences, where the probability of observing x given the ancestor c is expressed by $\phi_c(x) = P(x|c)$.

The last example illustrates a problem of random approximation: If $\phi_c(x)$ contains only few significant terms, i.e. c is rarely an ancestor of x and x', the number of samples to obtain a good estimate becomes very large. Thus, the approximated feature map $\overline{\phi}$ has to be reduced further.

Locality Sensitive Hashing

In locality sensitive hashing (LSH) the basic idea is to project data into a low-dimensional binary (Hamming) space. The values embedded in this space are generated by multiple hash

functions $h_1, \ldots, h_n$, from which each must satisfy the locality sensitive hashing property (Kulis and Grauman, 2012):

$$\Pr[h(\mathbf{x}_i) = h(\mathbf{x}_j)] = \text{sim}(\mathbf{x}_i, \mathbf{x}_j)$$

Here, $\text{sim}(\mathbf{x}, \mathbf{y}) \in [0, 1]$ is the similarity function to be approximated by LSH.

Charikar (2002) proposes a hash function that is based on rounding the product with the normal vector of a random hyperplane:

$$h_{\mathbf{v}} = \begin{cases} 1 & \text{if } \mathbf{v}^\intercal \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

The hyperplane defined by $\mathbf{v}$ is sampled from a multivariate Gaussian $\mathcal{N}(0, I)$ with I being the identity and 0 being the null matrix allowing to obtain Chernoff bounds.

A different perspective is given by Indyk and Motwani (1998): the concentration of measures due to the locality sensitive hashing property enables the dot product $\langle \phi(x), \phi(x') \rangle$ to be approximated by $\sum_{i=1}^{n} \langle v_i, \phi(x) \rangle \langle v_i, \phi(x') \rangle$. Both problem formulations show that LSH is again an application of random projections combined with hash functions.

Kernel Approximation

Shi et al. (2009) design a kernel based on pairwise independent hash functions $h : \mathcal{I} \to \{1, \ldots, n\}$ indexed by the set $\mathcal{I}$. Furthermore, they state $\phi_i(x)$, $i \in \mathcal{I}$ must be computed efficiently. Their definition of the hash kernel is then:

$$\overline{k}(x, x') = \langle \overline{\phi}(x), \overline{\phi}(x') \rangle \text{ with } \overline{\phi_j}(x) = \sum_{i \in \mathcal{J}; h(i) = j} \phi_i(x) \tag{2.17}$$

This means that all coordinates i of $\phi(x)$ for which the hash function $h(i)$ generates the same number j are mapped to the same coordinate $\overline{\phi_j}(x)$. We will apply this idea in the context of limiting the parameter space and reducing the feature space.

Information Loss

A natural concern about hashing in machine learning algorithms is the obvious fact that hash collisions harm performance. The birthday theorem helps to estimate how critical the problem is: assuming a hash function that maps into a space of size n, the probability of a collision reaches 0.5 with $\sqrt{n}$ features. Thus, although depending on the number of feature, collisions are relatively likely and each collision means information loss.

Having redundancy in features helps in many cases to avoid information loss. Shi et al. (2009) examine a form of *explicit* redundancy, where a feature is actually mapped to multiple values (buckets) in the space of size n. Their central observation is that information is lost only when all hashed features collide with other features. Thus, the logic approach would be to increase n by a factor c and then "duplicate" each feature c times.

However, the authors show that the probability of information loss can even go down when n is constant and c increases. For this to see, they take a random function mapping that maps l features c times into a space of size n and observe its behavior for all information loss functions and distributions on n features. Then the probability over the random function of having no information loss is bounded by:

$$1 - l[1 - (1 - c/n)^c + (lc/n)^c]$$

The authors give the example of $l = 10^5$ and $n = 10^8$: with standard hashing ($c = 1$), the bound is extremely high and estimations based on the birthday paradox show that the probability of having a collision is almost certain. If $c = 2$, the probability of having a collision drops to $1 - 0.596 = 0.404$, and with $c = 3$, it reaches a probability of $1 - 0.9883 = 0.0117$. The exact proof of the bound is given in Shi et al. (2009).

2.5 Feature Selection and Reduction

Feature selection is a very important task: the goal is to select subsets of features that are useful to build a good system. Thinking of the generalization capabilities of the learned system, it is not advisable to use all features because such systems tend to overfit (see, e.g. Murphy 2012). Occam's Razor, a general principle that has proven to be valid across many research fields, provides another justification for encouraging simplicity: "models should not be more complex than necessary to explain the data" (von Luxburg and Schölkopf, 2011).

Selection of useful feature subsets has recently become a very important research field in the area of machine learning. Modern applications like complex text processing or gene expression array analysis involve many variables. Their number usually exceeds hundreds of thousands and can even reach billions. Two developments are the motor for this :

- availability of large amounts of data
- increased computing performance

Although the latter seems to decrease the need of feature selection, in fact it is vice-versa: as pointed out in the previous section, the increased computational power nowadays enables methods and analyses that were not possible a few years ago. An example is the basic model (see Section 2.3) which cannot be directly applied to a full size vocabulary on today's computers, at least not in a non-distributed setting.

Forward and Backward Feature Selection

Not all features are equally important: features can be relevant or irrelevant with respect to a certain task. They can be dependent or independent, and sometimes the combination of multiple features is needed to achieve good results. The approach of ranking all potential

relevant variables and selecting the top-k is sometimes suboptimal, because even redundant variables can be important for building a certain classifier (Guyon and Elisseeff, 2003). The goal is to select *subsets of features* that are *useful* to build a good classifier. Traditionally, approaches for variable selection are divided into two groups:

- *Filters* select variables by ranking them using various correlation coefficient related metrics (Kononenko 1995 and Forman 2003). They are model independent and are usually applied as a preprocessing step to model selection and learning.
- *Wrappers* select subsets of variables according to the usefulness to the predictor. They treat the predictor as a black box and search for the best features in the space of all subsets. Since this problem is NP-hard, wrappers also suffer from computational complexity (Radivojac et al., 2004).

While filters are relatively easy to apply, they can cause problems. Intuitively, a correlation criteria seems to be a reasonable strategy to identify redundant variables, however, Guyon and Elisseeff (2003) construct multiple cases that illustrate the problems of this approach.

Figure 2.5 illustrates two cases where a variable is useless by itself but useful with others. Each case is represented by four squared areas, where the top-left and bottom-right squares are axis-projections of the datasets in the top-right and bottom-left square. Actually both projections represent the same dataset mirrored at the diagonal.

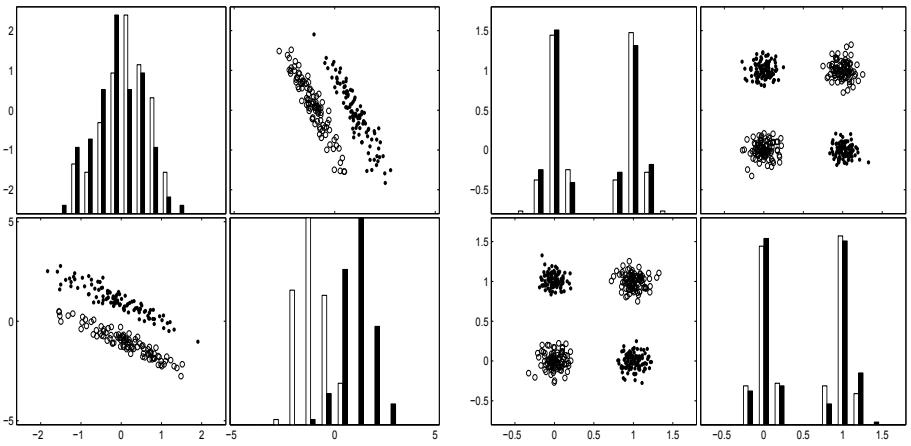

Figure 2.5: Two artificial examples where features are relevant that are individually irrelevant. From: Guyon and Elisseeff (2003).

In the left example of Figure 2.5 the means for one direction are clearly separated (bottom-right), for the other direction the datasets overlap completely (top-left), suggesting that the

latter variable is useless. However, a classifier is able to find a perfect separation in 2-dimensional space, whereas a 1-dimensional separation would be imperfect in this case. The right example is the famous XOR-case, where both variables seem to be completely useless, however, the combination enables classifiers with kernels to find a perfect separation.

There are more examples in the original paper, but these two are enough to illustrate the dangers of removing variables based on their usefulness. The possible price to pay is lost information, which can have a severe influence on quality of the predictor. The two examples are synthetic, so the impact on real-world tasks might be less. We will examine the effect of several basic filter methods in Chapters 4 and 5.

Wrappers, as defined by Guyon and Elisseeff (2003), have a different problem: computational complexity becomes infeasible if the feature space is large and the subset space is searched in a "brute force" manner, i.e. trying out all possible permutations of feature combinations. However, applying search strategies for feature selection can be beneficial in both reducing computational complexity and increasing generalization capability of the model, because limiting the number of parameters automatically reduces the danger of overfitting.

Embedded Methods

The wrapper algorithm treats the predictor as a black box. In contrast to this, the feature selection technique can be directly build into the predictor and be applied during training. Such approaches are called *embedded methods* as they are not wrappers by the given definition, although the result is comparable (Lal et al., 2006).

A common technique is to formulate an objective function consisting of two components that compete with each other :

1. reducing training error, i.e. the goodness of fit to be maximized
2. ensuring sparsity of the model, i.e. the number of variables to be minimized

This correspondence is formally derived and examined in detail by Weston et al. (2003). The authors propose a model were the ℓ_0-norm, i.e. the total number of variables, is approximated by a simple modification of the standard Support Vector Machine (Cortes and Vapnik, 1995) that applies a re-scaling step of the data during training.

Algorithms that iteratively increase the number of variables are said to apply forward feature selection. Respectively, algorithms that first consider the whole set of features and then iteratively decrease the number do backward feature selection. Both types of algorithms are usually implemented in a *greedy* fashion, i.e. they increase or decrease the number of variables in each step.

One very simple way to establish feature selection is to use an ℓ_1-regularizer for the learning objective (see Section 3.2). Chen et al. (2010) examined the effect of several variants of a linear model on different information retrieval tasks. Their best performing model uses a ℓ_1-regularizer in combination with a refitting of model weights. However, Bi

et al. (2003) showed that for certain applications it is sufficient to apply a ℓ_1-norm regularizer to set enough weights to zero to increase generalization capability of a model.

Limiting Parameter Space

Limiting feature space is an efficient way to reduce model complexity, but memory complexity can also be reduced more directly by limiting the number of variables of the model. Instead of allocating a possibly very large parameter array for the model in memory, the space requirements can be reduced by applying a hash function.

Hashing is a technique to map a large (possibly infinite) input set to a smaller (finite) output set. The underlying assumption is that the input space is relatively sparse and the finite number of elements can be mapped into a fixed size table. The mapping itself is calculated using a hash function, which reads an input key and outputs an index in the table. Ideally, the size of the table is equivalent to the number of elements and each position in the table holds exactly one element, meaning that the hash function is injective ("perfect hashing"). Usually, the mapping is one-way as multiple elements are mapped to the same position and additional overhead is required to keep track of keys and values, e.g. by using linked lists for hash table entries.

As explained in Section 2.3, the memory for storing the learned matrix or an equivalent weight vector is quadratically dependent on the vocabulary size. One way to reduce the memory requirement is to apply a variation of the kernel trick (see Section 2.2), where a high dimensional input vector $x \in \mathcal{X}$ is mapped to a lower dimensional feature space $\mathbb{R}^m$ with $\phi : \mathcal{X} \to \mathbb{R}^m$. Thus, the parameter vector lies in the m-dimensional space as opposed to the original input space. Weinberger et al. (2009) call this the *hashing trick*:

$$\overline{k}(x, x') = \langle \overline{\phi}(x), \overline{\phi}(x') \rangle \text{ with } \overline{\phi_j}(x) = \sum_{i \in \mathcal{J}; h(i)=j} \phi_i(x) \tag{2.18}$$

Thus, all coordinates i of $\phi(x)$ for which the hash function $h(i)$ generates the same number j are mapped to the same coordinate $\overline{\phi_j}(x)$. Shi et al. (2009) show that this kind of hashing preserves information in the same way as randomized projections for locality sensitive hashing (Gionis et al., 1999).

Filter Methods

For now, we have explained the idea behind embedded (wrapper) methods for feature selection. The other group of selection methods are filters, which are usually applied in a preprocessing step and are thus independent from the learning method. In our setup, the basic strategy is to filter the input space by applying certain methods, that usually aim at reducing the number of distinct words in the collection.

Limited Vocabulary

Limiting the vocabulary by keeping only the top-k most frequent words in a corpus is the most simple approach to reduce the input space. Throughout this thesis, we use the term *limited vocabulary* equivalently to *reduced dictionary*.

Although rare keywords, i.e. words with high TDIDF value, are especially useful for information retrieval tasks, experiments have shown that the information loss by removing each word below a cutoff count in a corpus is not as dramatic as it seems (see e.g. Section 5.2).

The works of Bai et al. (2010) and Chen et al. (2010) both examine basically the model presented in Section 2.3, and the authors use a limited vocabulary to control parameter space. Our main motivation is similar: controlling the size of the weight vector and thus reducing collisions of weights (see Section 3.1). Depending on the task, we experimented with reduced dictionary sizes of 10,000 (monolingual) and 30,000 (cross-language).

Please note that *vocabulary* in our sense differs from the linguistic definition: every string separated by spaces is counted, including punctuations, numbers, and special characters such as brackets or dashes.

Correlated Feature Hashing

The basics of feature hashing were explained in Section 2.4. This idea can decrease the memory requirements of the model substantially (See Section 3.1). The model can also be extended to features of query-document pairs in the following way (Bai et al., 2010):

$$\overline{\phi}_j(q,d) = \sum_{(s,t)\in\{1,\ldots,\mathcal{D}^2;h(s,t)=j} \phi_{s,t}(q,d) \tag{2.19}$$

Here, (s,t) is a single word pair between query and document and $\phi_{s,t}(\cdot)$ indexes this word pair, for example $\phi_{s,t}(\cdot) = \phi_{(s-1)\mathcal{D}+t}(\cdot)$. This basically means features share weights if their hash index is the same, i.e. $h(s,t) = h(m,n)$ implies $W_{st} = W_{mn}$. The problem of this approach is that collisions in the hash table are pseudo-random and result in sharing weights between pairs that usually have nothing in common.

At the same time, many works exploit collisions in hash tables as a useful property. For example, in *locality sensitive hashing* (Gionis et al., 1999) the central idea is to use a hash function, where similar elements are hashed to close buckets. As Charikar (2002) shows, there is no locality sensitive hash function corresponding to DICE coefficient (Dice, 1945) and Overlap, which are two measures for set similarity often used in information retrieval tasks. Regarding to his work, a very interesting approach is to actually learn such a hashing function with techniques for LSH, e.g. as described in Shakhnarovich et al. (2003).

In this work, we examine non-LSH mappings based on DICE coefficient. Bai et al. (2010) describe the construction algorithms for a mapping of words such that collisions occur for words with close meaning. This is accomplished by first generating a list of $\mathcal{D}$ words sorted by frequency. Then for each word $i = 1,\ldots,\mathcal{D}$ a variant of the DICE coefficient for words as defined by Smadja et al. (1996) is calculated for the whole reduced vocabulary $j = 1,\ldots,\mathcal{F}$:

$$\text{DICE}(i,j) = \frac{2 \cdot \text{cooc}(i,j)}{\text{occur}(i) + \text{occur}(j)} \tag{2.20}$$

Here, $\text{occur}(i)$ is the number of occurrences of word i in the whole collection. The expression $\text{cooc}(i,j)$ is the number of times word i occurs together with word j. However, it is not clear at which level (e.g. sentence or document) co-occurrence is evaluated. It is reasonable to think of co-occurrence as a binary predicate: either two words occur together at a certain level or they do not. A question arises if words occur multiple times within the area of interest: how should multiple co-occurrences be handled in the most efficient way?

Melamed (1998) analyses different counting methods for the task of word co-occurrence in parallel texts. Although the setting is different, the two counting schemes he outlines are also applicable in our case:

$$\text{cooc}(u,v) = \min(e(u), f(v)) \tag{2.21}$$
$$\text{cooc}(u,v) = \max(e(u), f(v)) \tag{2.22}$$

The intuition behind Equation 2.21 is that co-occurrence is the number of times complete pairs between u and v can be extracted, while Equation 2.22 expresses how often a term u was observed in the vicinity of term v. These counting types can be understood as creating unions and intersections on multisets, while the binary variant initially explained works on normal sets.

This gives a list of $\mathcal{F}$ co-occurring words for a given word. By sorting the list by descending DICE score, the list contains a mapping of a word to correlated words where more correlated entries are higher in the list. Following Bai et al. (2010), we define $S_p(i) \in \{1, \ldots, \mathcal{F}\}$ as the index of the pth largest DICE score $\text{DICE}(i, S_p(i))$. Applying 2.19 results in the mapping:

$$h(m,n) = (S_1(m) - 1)\mathcal{F} + S_1(n) \tag{2.23}$$

Or, in other words, a pair $(m,n) \in \mathcal{D}^2$ is mapped into a matrix of size $\mathcal{F}^2$.

This approach is called *correlated feature hashing* and it effectively reduces the feature space from $\mathcal{D}^2$ to $\mathcal{F}^2$. A very useful property of this method is that $\mathcal{F}$ can be adjusted to the needs for complexity and to available resources.

Although this method seems to delete less information than limiting the vocabulary, the loss of content depends on the mapping. If this mapping is computed on a smaller corpus, singletons[3] are likely to have co-occurring words with the same DICE score, so there exists a "plateau" of identical scores at the top. According to Equation 2.23, the algorithm will select only the first word while the order depends on the data structure implemented to organize the mapping in memory.

[3]Words that occur only once in the whole corpus.

As shown in Section 2.4, Shi et al. (2009) suggested to approximate a kernel by using a sum of multiple feature maps based on hash functions as in 2.17 for the dot-product. They prove that this construction can actively reduce the probability of information loss that occurs when entries in the hash table collide (see Section 2.4).

Given the definition 2.18 and the mapping presented in 2.23, it is straightforward to extend this idea to the feature map construction of hash kernels. Instead of selecting the single word with the highest DICE-score and apply this to Equation 2.19, one would use the top-k words to create k hash functions (mappings):

$$\overline{\phi_j}(q,d) = \sum_{p=1,\ldots,k} \phi_{s,t}(q,d) \tag{2.24}$$
$$(s,t) \in \{1,\ldots,\mathcal{D}^2 : h_p(s,t) = j$$

Analogous to Equation 2.23, the pth hash function is then defined as:

$$h(m,n) = (S_p(m) - 1)\mathcal{F} + S_p(n)$$

Bai et al. (2010) call this approach *correlated feature hashing with multiple binning* and it has the same reduction capabilities as correlated feature hashing. Additionally, it reduces the problem of information loss by hash collisions by implementing the hash kernel idea explained in Section 2.4.

Final Word on this Section

We end this section on methods for feature selection and reduction and the ideas we plan to apply by pointing out that we were only able to present a very limited section of this field. Feature selection is an active research area where many important developments happen at the moment.

This introduction in aimed at giving an overview on the methods to explain the importance of this difficult task and to better understand the problems involved in it.

Chapter 3

Training and Test Framework

This chapter contains descriptions of the technical solutions and algorithms involved in our work. We will first explain in Section 3.1 what the challenges are regarding *complexity* during the training and testing phase, and how we use modern parallelization techniques in our setup. Section 3.2 contains descriptions of our *implementation* to create and preprocess data in the various ways we need for learning and evaluation. We also explain the tool we use and how it can be applied to our problem. We finalize this chapter with Section 3.3, where we describe different retrieval metrics and the significance test we use during *evaluation*.

3.1 Complexity

Using quadratic word features is a special challenge for both training and testing algorithms. According to the law of Herdan-Heaps (Heaps, 1978), the number of words in any language grows unbounded with the size of the text collection:

$$V_R(n) = Kn^{\beta} \tag{3.1}$$

Here, V_R is the number of distinct words in the vocabulary, n is the size of the collection, and K and β are values empirically estimated.

Following this law, even in medium sized corpora the vocabulary can easily exceed a million entries, making the creation of quadratic word features problematic: for example, Bai et al. (2010) counted 2.5 million words in $1,828,645$ English articles of their Wikipedia corpus, resulting in $6.25 \cdot 10^{12}$ entries in the parameter vector. Assuming single precision floating point numbers (4 bytes), this would consume 25 Terabytes of random access memory (RAM), and twice as much in the case of double precision numbers. Although this amount of memory is available for distributed systems, the nature of the learning task normally requires the weight vector to be kept completely in memory on a single machine.

However, the hashing trick (see Equation 2.18) enables in principle an arbitrary sized parameter vector, where the implied information loss can be controlled by the overload

Type	latency	order of size [bytes]
Processor registers	≈ 1	10^2
Level 1 cache	few cycles	10^4
Level 2 cache	10 to 50 cycles	10^5
Level 3 cache	higher than L2	10^6
Main memory	hundreds of cycles	10^9
Disk storage	millions of cycles	10^{12}

Table 3.1: Comparison of latency for different types of computer memory.

factor of the virtual hash table. A state-of-the-art implementation of a hash function is MurmurHash[1], of which different versions for various bit sizes (32/64/128) exist.

A reasonable compromise between speed and size is the version that calculates 32-bit wide hashes, because the application interprets these hash values as addresses of parameters. The next step with 64 bits will need twice the memory for each pointer and will lead to significantly lower computing performance, while a certain amount of bits are mostly useless[2]. Using 32 bits results in $2^{32} \approx 4.295 \cdot 10^9$ entries in the hash table. In the above example, this would result in an overload factor of approximately $1,455$ for the hash table. As this is the amount of average collisions per bucket, this setting would make it impossible for the learner to extract useful information.

Thus, the main strategy must be reducing the feature space while trading-off the following two aspects:

- Controlling model complexity, i.e. keeping the overload factor low
- Controlling feature representation, i.e. filtering with minimized information loss

A different aspect of memory complexity is illustrated in Table 3.1. The strong connection between memory size and access performance shows that there is another reason to keep a model small: going from L1-cache to main memory increases access latency by several orders of magnitude, so it is important to keep an eye on data structures and model sizes to achieve a system that performs well with respect to computation time. However, optimization of algorithms on the hardware-level is a difficult, time-consuming, and very specialized task. In the end it is unrewarding, because the better the optimization, the more it is specialized strictly on a single specific task.

It is thus necessary to establish a feeling about trading off the amount of time for analyzing a problem and optimizing the code against the time won by this optimization step. Having the memory wall (see Table 3.1) in mind, it can often help to optimize execution time by very simple yet effective measures. Keeping often accessed parameters within a certain address

[1] `http://murmurhash.googlepages.com/`, accessed 06.09.2013

[2] 40 bits are enough to address more than 10^{12} buckets, an amount that is rarely used up even on bigger compute servers.

range will speed up frequent accesses by many orders of magnitude. Moving from hash tables to vectors whenever possible can have a great impact on execution time because of built-in memory cache organization and prefetching strategies.

Training and Test

The main factors that influence the resource requirements for the basic model 2.13 are data size and the size of the parameter matrix. The learning strategy we follow is *online learning*, so data is generally processed sequentially. For testing, we evaluate the data document by document against all queries and calculate scores on the different pairs.

Training

Our approach relies completely on online learning, so all data is kept on disk and is processed sequentially. As a consequence, we move the process of sampling random data pairs from the learning tool to a utility that creates the training data on disk.

This strategy has several benefits: we heavily decrease the resource requirements during learning. By avoiding to load large amounts of training data into RAM and generate random pairs, we can use the additional memory resources to increase model complexity and/or learn more models in parallel. If the training data resides as files on disk, comparing different settings is very easy: many models running under different configurations can work on the same data in parallel. The SGD-approach means that besides the model (i.e. the parameter vector) a learner only needs to keep a pointer to the position in file containing the training data. Finally, using a common training file ensures that all instances of the tool work on the same "randomized" training data, so the results are better comparable even for smaller training sets. The only drawback of this method is that the off-line data creation gets more complex, bu this is a one-time action.

A critical aspect not mentioned yet is data creation for the quadratic model. Even short vectors with 200 features will grow to a size of 40,000 after quadratic expansion. A normal document can easily contain 1,000 words, which will expand quadratically to 1,000,000 features. Thus, it is not desirable to expand the vectors on disk. Fortunately, the learning tool described in Section 3.2 addresses this problem by doing the quadratic expansion on-the-fly.

To finally get the training data on disk, we need to establish a construction scheme that simulates the SGD-idea described in Algorithm 1. A brief overview of the required steps to create the appropriate training instances is given in Section 3.2.

Test

Testing implies basically the same computational problems as training complexity with the difference that we do not need to simulate a random selection of pairs as in SGD. The naïve

approach to generate testing data for our evaluation would be to create all possible pairs of queries and documents on disk and let the algorithm work his way through the list.

This is not desirable, because we would have to create a quadratic number of possible pairs on the filesystem. This quickly reaches sizes that are difficult to handle. The solution is to use the on-the-fly approach for the test data: the content of this data are queries and documents, combined in a way that each query-document pair is evaluated by our model and returns a score. It is thus much easier to take the list of query vectors and the list of document vectors and create pairs on-line during evaluation.

A problem might arise when the document and the query collection do not fit into memory, but even then the idea can be combined with reading queries and documents from disk. This method has another positive aspect: it can be easily parallelized on distributed systems, as the MapReduce approach described at the end of the section shows.

Parallelization Strategies

Modern algorithms and implementations have to be analyzed with respect to their scalability. The previous section showed that there are challenges in memory and computational complexity involved in our approach. This section will give a brief technical overview about the main architectures for parallelized algorithms and which techniques can be applied in our implementation.

We will only talk about one class of computer architectures defined in the taxonomy by Flynn (1972), *multiple-instruction-multiple-data (MIMD)* machines. There is one more classification that is interesting with respect to modern computer hardware, the *single-instruction-multiple-data (SIMD)* class. SIMD machines are today mostly graphics processing units (GPU), sometimes specified as general purpose GPUs (GPGPU) to emphasize their use across graphics applications. Although they provide enormous computing power, their application was out of the scope of this work.

Shared Memory Systems

Shared memory (SHM) systems are the most common group of computers today. Basically every computer today has a Processor (CPU) with multiple cores that are able to execute programs truly in parallel.

The memory logic of such a system is illustrated in Figure 3.2 (left). Each processor (P) has access to all memory (M) available in the system. Access to a memory location is *cached*, i.e. copied to a very fast cache-memory (C), that speeds up repeated access to the same location by multiple orders of magnitude (see Table 3.1). This usually works well because program and data have a high degree of *locality*, which means they access the same program parts or data structures many times (e.g. running a loop or iterating over an array) .

For SHM, this strategy becomes problematic when data is written to memory, because existing copies of the changed memory locations in all the caches must be invalidated. This is accomplished by protocols that help monitoring memory changes and invalidating cache

entries if necessary. However, when multiple CPUs write to the same memory location, the efficiency of the cache, i.e. the factor by which the cache is faster than main memory, converges to 1 (*cache thrashing*). This is one of the problems that define the physical boundaries for building SHM systems. The largest SHM systems today combine up to 128 cores, but the price/performance ratio is 20 times higher than for low-end server hardware (Barroso and Hölzle, 2009, p. 32).

SHM systems are very easy to program, because they provide one shared view on the data. For the machine learning application we use, this does not help much: if we update the weight vector in parallel, cache thrashing occurs and at some point the parallel learning process will be slower than running the process on a single core.

There are two small applications where the SHM approach is actively used by us: First, we use OpenMP v3.0 (OpenMP Architecture Review Board, 2008) to sample training data in parallel. Reading from memory does not imply any cache coherence problems, and OpenMP is very simple to use: the programmer basically writes a sequential program and marks certain sections of the code which are to be parallelized, and the compiler does the rest. However, the result is rarely optimal and hand-crafted code using low-level functions of the PThread library (Narlikar and Blelloch, 1998) can perform much better, at the cost of non-portability and increased effort for development and debugging. Second, the task of data preparation, especially reading from disk and parsing the data, can be moved to a separate process. This does not give a huge increase in processing performance, but the gain is constant and reliable.

A third application of the SHM approach exists within the Hadoop framework introduced in Section 3.2. As the SHM is used for faster intra-process communication within the Hadoop framework, we do not deliberately make use of SHM there.

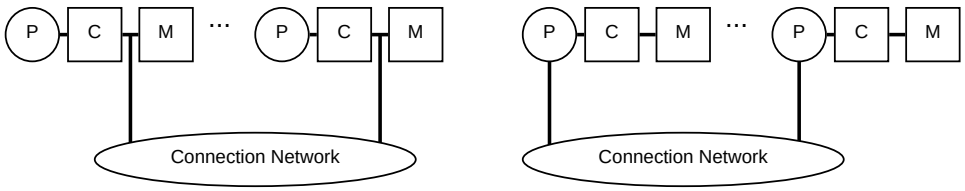

Figure 3.1: MIMD computing architectures: shared memory (left) and distributed memory systems (right).

Distributed Memory Systems

The other strategy to build MIMD systems is to follow a distributed memory (DM) approach. The basic idea is illustrated in Figure 3.1 (right). The main difference to the SHM idea is that each computer has its own private memory so processors in the network cannot access other processor's data. Working only on private memory, the problem of cache coherence is

non-existent because it cannot happen that a CPU invalidates the cache contents of another CPU. The drawback is that there is no sharing of data – non-private data must be requested and received via messaging, which is generally very slow, at least several orders of magnitude slower than standard memory access.

A very crucial problem in this programming framework is that programs can easily run into race conditions, e.g. one processor 1 requests a message from processor 2, while processor 2 requests a message from processor 1. The system is stalled, the two nodes are in a "deadlock". Thus, programming DM systems requires special algorithms and debugging is extremely difficult.

The standard framework for programming DM systems is the *Message Passing Interface (MPI)* (Message Passing Interface Forum, 2009). It provides a system for writing Fortran77 and C programs that use messages for interprocessor communication. MPI was invented in the 1990s and is today widely used for programming distributed memory systems.

The underlying strategy for writing MPI programs is different to program SHM systems. The cost for data exchange is often the limiting factor, so data has to be distributed among the nodes in a clever way. On one hand, nodes should be able to work independently as long as possible, on the other hand data exchanges should be minimized. This is often a trade-off situation and the optimal settings depend on the type and order of operations applied to the data.

MapReduce

In recent years, other programming frameworks such as MapReduce (Dean and Ghemawat, 2004) and DryadLinq (Yu et al., 2008) became very popular, because they move the complexity involved in writing correct parallel code into a different abstraction layer, so the programmer does not need to think about concurrency, data consistency, race conditions, or data distribution.

The basic idea of MapReduce is related to the divide-and-conquer approach known from other well-known algorithms like Quicksort. However, the idea of *recursively* dividing a bigger problem into smaller subproblems until they are easily solvable is absent here. What is similar is the strategy to distribute the data to make a big problem simpler and to aggregate partial results.

MapReduce was built with processing large amounts of data in mind. In such, the whole framework is optimized for applications that apply simple operations on large amounts of data. It is suboptimal for applications where data has to be processed multiple times using different operations, e.g. numerical calculations like matrix-matrix-multiplications.

In the MapReduce framework, the programmer typically specifies 2 functions, *map* and *reduce*. The *Mapper* function usually works on partitions of data and emits partial results, while the *Reducer* function collects partial results and emits (part of) the final result. Defining these two functions is basically all that has to be done from a programming point of work, the framework is responsible for distributing the data, aggregating results, synchronization, etc. As a side note, there are several optimizations possibilities by defining additional func-

tional elements called *Combiners* and *Partitioners*, that can speed up a map-reduce cycle significantly. And further, MapReduce programming is especially about constructing clever data structures that exploit the basic functionality of the framework. More details about this can be found in the excellent textbook by Lin and Dyer (2010).

We will not go into the details, but it is interesting to see how our evaluation of retrieval is implemented on MapReduce, although MapReduce is in general not suited for retrieval tasks (Lin and Dyer, 2010, p. 86). The main difference between a real-world retrieval application and evaluating retrieval systems is that in the former case, a person enters a query and expects the retrieved results *as fast as possible*, while in the latter case a system processes a list of queries and returns the results (e.g. documents and scores) for each query. In addition to that, our system "searches" a collection by creating scores for all possible query-document pairs and sorting this list descending by score.

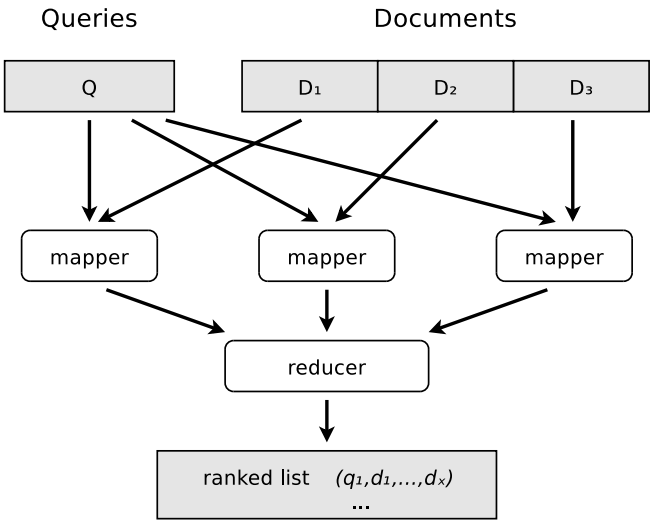

Figure 3.2: MapReduce application for the "embarrassingly parallelizable" problem of retrieval system evaluation.

Such tasks are called "embarrassingly parallelizable" because there is no data dependency between the processes. The evaluation pipeline for our retrieval system works as illustrated in Figure 3.2: the set of queries Q is loaded by every Mapper, and the collection set of documents $D_1, \ldots, D_n$ is distributed among n Mappers. Each Mapper evaluates every query against the partition of documents it has read into memory and emits the query-ID, document-ID, and score to the MapReduce system. Then, the framework sends all retrieval results belonging to one query to the same Reducer, where the retrieval results are sorted and finally written to disk.

For our retrieval experiments we use a MapReduce cluster of 30 nodes. Every node contains 2 CPUs with 4 cores each clocked at 2.4 GHz, 24 GB random access memory and

approximately 2.7 TB for the Hadoop filesystem HDFS. In total there are 240 cores, 720 GB RAM and 87.4 TB available for data processing.

3.2 Implementation

The implementation of the system first requires preprocessing of the data to generate the representation suitable for our information retrieval task. There is no need to create inverted indexes for retrieval, because the system evaluates all documents as possible candidates for the search query. Preprocessing involves applying the filter methods described in Section 2.5.

The second step is to generate data for training in a format that contains not only the query, but also the pairwise difference of relevant and irrelevant documents (see Section 2.2). This training data is then fed to the learning tool, which generates a model that captures the patterns hidden inside the data (see Figure 2.4).

The models are tested on a Hadoop[3]-cluster, the JAVA[4]-based Open-Source version of a MapReduce framework implementation. More details on the retrieval evaluation pipeline were explained in the previous section.

Creating Data

To extract TFIDF-vectors from raw text data, we applied several tools written by Felix Hieber for a project with Laura Jehl on CLIR using SMT models[5] based on the cdec-framework (Dyer et al., 2010). The TFIDF-vectors are then converted into a format that can be read by our learning tool.

Either during vector creation or during format conversion we apply different filter methods which are described in detail in the corresponding sections of Chapters 4 and 5.

Test Data

Based on this vector representation, we create the testing data on-line. Even for small corpora expanding data for testing will become cumbersome, because it would be required to explicitly create all possible pairs between queries and documents. For the test part of the 20NG corpus described in Chapter 4, this means creating $7532^2 = 56,731,024$ lines of data, and each line will on average contain approximately $290.6^2 \approx 84,451$ entries, i.e. a word and a value.

If the expansion of vectors is done on-line, we only need the basic set of vectors in the appropriate format. In the example above, the query and document files both have 7532 lines of data with 290.6 entries on average.

[3] `http://hadoop.apache.org/`, accessed 01.08.2013

[4] `http://www.java.com/`, accessed 01.08.2013

[5] `https://github.com/felleh/cdec/`, accessed 06.09.2013

Training Data

For training data, however, on-line data creation was initially applied during learning but later removed due to performance issues. The structure of vectors for training is more complex than for test. The main reason is that the basic representation contains query vectors and differences of vectors of relevant and irrelevant documents, making on-line creation very memory consuming. To create pair-wise vectors for SGD-based learning algorithms, it is necessary to keep *all data* in memory, i.e. the query and document representations, and the relations between queries and documents.

Creation of a single training instance follows these steps:

1. A query vector $\mathbf{q}$ is selected from the query pool.
2. A relevant document $\mathbf{d}^+$ is selected from the list of relevant documents for the query.
3. An irrelevant document $\mathbf{d}^-$ is selected that is not on the list of relevant documents.
4. The three vectors are combined to build the pair $(\mathbf{q}, \mathbf{d}^+ - \mathbf{d}^-)$.

Another reason for moving from on-line to off-line training data besides memory considerations is that once created on disk, such data can be used by different learners under different settings simultaneously. Although training data can become large, it only has to be created once.

Preprocessing

Feature reduction techniques such as limiting vocabulary and correlated feature hashing had to be applied as a preprocessing step before TFIDF-creation, sparse vector conversion and generation of training and test data takes place.

Limiting the vocabulary is a simple task. The first step is to go over the whole corpus and create statistics on word counts. This list is sorted descending by count and then cut off at the kth entry, resulting in a list of size k of the most frequent words. Finally, the corpus is filtered by removing all words not present in this list.

The method of *correlated feature hashing*, requires a deeper analysis of the data. We propose a fast method based on inverted indexes, that can be easily extended to a MapReduce setting. At the moment, our implementation is single threaded, but fast enough to process millions of documents within minutes on standard hardware.

The implementation of the algorithm that creates a mapping of a given word i to m words from a limited vocabulary of size k works as follows:

1. Go through all documents and create an inverted index (lists of documents for each word) as well as total counts for each word i.
2. Using the total counts for each i, create a limited vocabulary V' of top-k words.

3. Iterate over the inverted index and go through each document, creating co-occurrence counts between i and each document word $j \in V'$.

4. For each word j, calculate DICE-scores according to Equation 2.20 and sort the final list descending by DICE.

5. The final mapping for a word i are the top-m words j from the list created under (4).

The current implementation consumes a lot of memory and there is much room for improvement, e.g. the inverted index can be represented more efficiently in memory using compression codes on gaps. However, this implementation is absolutely sufficient for the tasks we examine. Lin and Dyer (2010) describe various aspects of the construction of an inverted index in MapReduce, and by correctly applying typical design patterns for Hadoop programming, the idea described above can be scaled up to work on much larger corpora.

Learning Tools

We started our work with a learning tool developed by Fendrich (2012). His tool Sol[6] is a toolkit written in C++ for fast learning of linear models using an SGD approach. The first experiments with the basic model were conducted using this tool. Although initial experiments on small data were promising, we were struck by the impact on memory complexity for the quadratic model as soon as we moved to real-world data.

During the implementation of countermeasures like singular value decomposition (SVD) to learn on decomposed low-rank matrices, we came across a tool named *Vowpal Wabbit*[7] that already contained most of the things we wanted to implement and which was already at a very high development stage at that time.

Vowpal Wabbit (VW) is a sophisticated learning tool that started as a project at Yahoo! Research[8] and is now continuing at Microsoft Research[9]. The project "Vowpal Wabbit (Fast Learning)" is ongoing work developed and supported by many contributers. It has several outstanding features, from which we name a few here.

VW implements several very fast learning algorithms for use with large datasets. In addition to standard gradient descent it can also optimize using the Conjugate Gradient or the L-BFGS method (Nocedal and Wright, 2006), a quasi-Newton approach that can be applied when the Hessian is too large to be kept in memory. In addition to that, VW can learn on mini batches.

VW does on-line learning, meaning that it can work on unbounded data sizes: practically, we use data from hard disk, which is our limiting factor, but theoretically VW can work on infinite data streams. The weight vector is kept in memory employing the hashing trick

[6] `https://github.com/sfendrich/Sol`, accessed 03.09.2013
[7] `http://hunch.net/~vw/`, accessed 06.09.2013
[8] `http://labs.yahoo.com/`, accessed 06.09.2013
[9] `http://research.microsoft.com/`, accessed 06.09.2013

(see Equation 2.18), so the number of features is in principle unbounded. The size of the parameter space can be defined by exponents to the basis 2 ($= 2^x$ variables).

However, the best technical feature for us is the internal quadratic expansion of namespaces. VW employs namespaces in the data format, and a command line switch during learning enables on-line creation of quadratic features: this means, every feature in one namespace is combined with every other features in the other namespace. Apparently, this idea is so successful that the authors recently added a cubic switch to create cubic features of three namespaces on-the-fly.

Constantly new features are integrated into the VW source. Besides many different optimization algorithms VW provides an implementation for sparse singular velue decomposition (SVD) based on Koren et al. (2009), a module to calculate Latent Dirichlet Allocation (Blei et al., 2003), a clustering mode for Hadoop and MPI etc.

Finally, as VW brings together much more than all the components and techniques we ever wanted, we decided to switch to Vowpal Wabbit as the general learning tool for all future experiments of this work.

3.3 Evaluation

Evaluation of an IR system means running the system against a set of queries and documents of known relevance and comparing the retrieved result with the expected result. The best system is the one that gets the highest scores, while the score expresses how well a system reproduces the relevance judgments for known query-document pairs.

Metrics

Information Retrieval metrics measure the effectiveness how relevant documents are retrieved by a system. There are several metrics that became a quasi-standard over the years, pushed especially by the annual Text REtreival Conference[10] (TREC). TREC is a conference that provides tracks, i.e. challenges on test problems where participating groups compete against others on given datasets.

Representation of a metric aims at being simple and informative to enable objective comparison of the retrieval performance of systems. However, different metrics address different aspects of retrieval problems. In the following, we explain the metrics we use for the experiments in Chapters 4 and 5, and why they are a good choice in our opinion.

Precision and Recall

In the field of information retrieval, the metrics precision and recall are calculated on sets of retrieved and relevant documents. In words, precision is the fraction of documents that

[10] `http://trec.nist.gov/`, accessed 09.09.2013

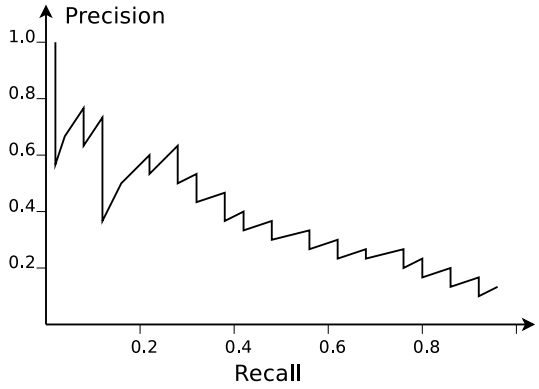

Figure 3.3: A precision-recall curve with the classic sawtooth shape.

were correctly retrieved ("how many are correct"), while recall is the fraction of documents that were successfully retrieved ("how many were found").

Let $\mathcal{R}$ be the set of relevant documents and $\mathcal{S}$ be the set of all retrieved documents for a given query. Then precision and recall can be defined in set notation as:

$$Precision = \frac{|\{\mathcal{R}\} \cap \{\mathcal{S}\}|}{|\{\mathcal{S}\}|}$$

$$Recall = \frac{|\{\mathcal{R}\} \cap \{\mathcal{S}\}|}{|\{\mathcal{R}\}|}$$

As both metrics are set-based, the calculation is done on limited subsets, e.g. the top-k results of retrieval results.

If precision and recall are plotted in 2-dimensions as in Figure 3.3, the graph shows a typical sawtooth shape: if the $(k+1)th$ retrieved document is not relevant according to the search, then recall will stay the same but precision drops, the curve goes straight down. If the $(k+1)th$ retrieved document is relevant, then precision and recall increase and the curve goes diagonally up. The sawtooth shape is sometimes smoothed by plotting an interpolated precision, i.e. the precision at a certain recall level r is the highest precision for any recall level $r' \geq r$ (Manning et al., 2008).

Although the information of a complete precision-recall-curve is very informative, the curve is difficult to interpret and there are cases where it is better to have a single number, for example when comparing several systems among each other.

The most simple way to reduce the graph to a single number is to calculate precision until a certain cutoff position k in the retrieved list. Let $L = (r_1, \ldots, r_n)$ be a list with r_m being a binary value indicating the relevance of a document at position m. Then the value

P@k is defined as:

$$\text{P@}k(L) = \frac{1}{k}\sum_{i=1}^{k} r_i \tag{3.2}$$

The same can be calculated for recall, expressing the recall at a cutoff position k in L:

$$\text{R@}k(L) = \frac{1}{|\mathcal{R}|}\sum_{i=1}^{k} r_i \tag{3.3}$$

While the precision metric from Equation 3.2 is frequently used, recall values based on Equation 3.3 are rare.

Mean Average Precision

The metrics P@k and R@k are much easier to compare, but they still contain a parameter k and it depends on the task (and probably its interpretation) which k should be applied. Another idea to reduce the graph in Figure 3.3 to a single number is to calculate the area under the precision-recall curve by integrating precision over the recall:

$$\text{AP}(q) = \int_0^1 p_q(r)\ \mathrm{d}r = \frac{1}{|\mathcal{R}_q|}\sum_{i=1}^{|\{\mathcal{S}_q\}|}(P_q(i)\cdot r_i)$$

Here, $p_q(r)$ is the precision-recall function for a single query q, and $P_q(k)$ is the precision for query q at cutoff position k (P@k). So, the average precision is the area of the precision-recall curve for a certain information need q.

If the average precision is calculated for all queries and the average precision values are averaged by taking the arithmetic mean, then the result is called *mean average precision (MAP)*:

$$\text{MAP}(Q) = \frac{1}{|Q|}\sum_{q=1}^{|Q|}\text{AP}(q)$$

For evaluation of our experiments in Chapters 4 and 5, we use the `trec_eval`-script (v8.1) provided by the TREC evaluation campaign[11] to generate MAP scores. Although there are many other metrics provided by this script, we will use only MAP because it is widely accepted and scores can be compared to other works.

A short note on the range of values: MAP is defined such that it returns a value between 1.0 (perfect ranking) and 0 (completely wrong) and we use this range for reporting MAP scores. However, when we speak of *MAP points*, we mean "percent" in the MAP range, i.e. 1 MAP point equals to 0.01 MAP score.

[11] `http://trec.nist.gov/trec_eval`, accessed 10.09.2013

Other Metrics

In recent years, many other IR metrics have been developed, e.g. "Receiver Operating Characteristics" (ROC), which creates a graph of the true positive rate against the false positive rate, or "Normalized Discounted Cumulative Gain" (NDCG), a measure that can be applied to non-binary relevance cases, just to name two that are frequently in use. NDCG is the "cumulative information gain" of a set of queries and the returned list of top-k results of different relevance, and then normalized so the optimal ranking is 1.0. Evaluating different relevance levels is especially useful for testing web search applications: the motivation is that a user expects to receive the most relevant entries within the first top-k results, preferable on the first page.

As for MAP and NDCG, the underlying strategy is focused on precision, because in many applications of information retrieval, the user wants the most relevant results to appear as high as possible, e.g. on the first search results page or among the top-10 entries.

However, there are cases where the user is willing to review more than only the top-10 results, because it is important to find as many relevant documents as possible. Even for these recall-oriented tasks standard precision-based IR measures are still frequently used.

One common recall-oriented task is patent retrieval, where the user searches for patents that are relevant to a given patent (see Chapter 5, Section 5.1). The reliability of these searches are of high economic importance, because missing a relevant patent can have a high impact on validity of the given patent, and consequently on the inventor, the applicant, and the attorney. Thus, a user of a patent search engine is willing to add extra effort to evaluate a longer list, e.g. by grouping, sorting, or structuring the results in a post-processing step.

A recall oriented measure that was developed especially for evaluating patent retrieval systems is the *patent retrieval evaluation score* (PRES) (Magdy and Jones, 2010). Its idea is based on the *normalized recall* R_{norm}, which measures basically the fraction between the current system's performance against the best case in terms of recall.

Let r_i be the rank at which the ith document is retrieved, n being the number of relevant documents and N the collection size, then the normalized recall is defined as:

$$R_{norm} = \frac{A_2}{A_1 + A_2} = 1 - \frac{\sum r_i - \sum i}{n(N-n)} \tag{3.4}$$

Figure 3.4 illustrates the idea behind R_{norm}, which is the area between the actual and worst case (A_2) divided by the area between the best and the worst case ($A_1 + A_2$).

The problem with R_{norm} is that it is calculated on the whole set collection. With millions of documents in the retrieved list, the calculation will become very expensive. PRES addresses this issue by defining a different procedure for the worst case: the number of relevant documents is assumed to be retrieved just after the number of documents checked by the user, i.e. after position N_{max}. The idea behind this strategy is that any document after N_{max} is missed by the user, and if all relevant documents are located after N_{max}, the score will be zero.

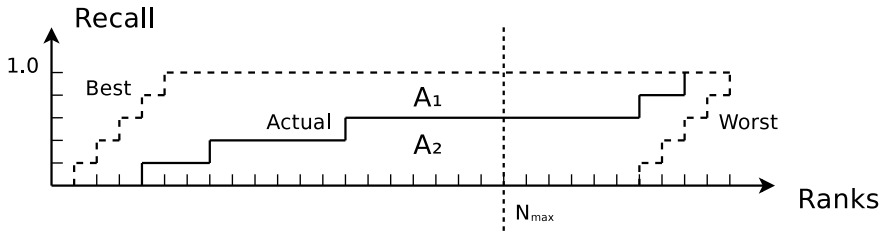

Figure 3.4: Connection between normalized recall and PRES metric for an example with 5 relevant documents. Combination of two illustrations from Magdy and Jones (2010).

Any document not retrieved before N_{max} will contribute to the newly defined worst case. This is achieved by replacing N in Equation 3.4 with $N_{max} + n$, where n is the number of relevant documents. After simplifying the expression the final score for PRES is:

$$\text{PRES} = 1 - \frac{1}{N_{max}} \left(\frac{\sum r_i}{n} - \frac{n+1}{2} \right) \tag{3.5}$$

Equation 3.5 shows that PRES is a function of the recall of the system, the ranking of retrieved results and the number of documents checked by the user. This combination makes it an excellent choice for an additional metric for our experiments on patent data in Chapter 5.

A detailed analysis of the PRES metric on 48 runs from the CLEF-IP 2009 patent retrieval track (see Peters et al., 2010) is given in Magdy and Jones (2010). In their experiments, the authors demonstrate that PRES agrees with MAP and Recall in more than 70% on a pairwise comparison of the submissions, with a high correlation among the low and the high ranked positions.

A final note on the range of values: similar to MAP, PRES returns a value between 1.0 for "perfect ranking" and 0 for "completely wrong" (everything ranked below N_{max}). We use this range for reporting scores, but when we mention *PRES points*, we mean "percent" in the PRES range, i.e. 1 PRES point equals to 0.01 PRES score.

Significance Testing

A common strategy to find good parameters is to run a systems on the same data under different parameter settings and analyze the score changes. The system with the highest score is believed to have the better setting. This conclusion is often unquestioned because metrics seem to be objective measures, but there is more to think of if one wants to draw *reliable* conclusions based on score values.

It is necessary to estimate if a score change is significant such that a difference in systems can be presumed, or is merely a result of random variations produced by the same system.

The test that gives an answer to this question is called *significance test* and it is based on the method of hypothesis testing.

Given two systems X and Y with, for example, N topics, and two scores calculated on the systems μ_{OLD} and μ_{NEW}, the method of hypothesis testing is as follows (Cohen, 1995): 1) Formulate a null hypothesis H_0, typically saying that there is no difference between the systems X and Y. 2) Create all possible 2^N permutations between system X and Y, calculate the difference of scores and count how often it is greater than the difference of the observed scores $\mu_{OLD} - \mu_{NEW}$. 3) The count divided by the number of permutations is the *achieved significance level* or *p-value*[12]. 4) The p-value is the probability of achieving the observed result by chance *under* the null hypothesis H_0. In other words, p is the probability of incorrectly rejecting H_0 that the systems X and Y are equal. Thus, a low p-value indicates that two systems are different and the change in score is significant.

However, there is a problem with the method described above. In IR systems, the topics are usually queries for which retrieval accuracy is calculated. Even for only 50 queries a system had to evaluate $2^{50} \approx 10^{30}$ permutations of data to calculate significance between only two candidate systems or settings. Such a test that takes every possible permutation into account is called an *exact randomization test*. Fortunately, it is in general not necessary to do such an exhaustive analysis. The Monte-Carlo-method (Metropolis and Ulam, 1949) is a very efficient way to approximate systems that are too complex to be analyzed exactly. The idea behind *Monte-Carlo approximation* is that it is sufficient to sample from the set of all permutations and still achieve a relatively low standard error on the result.

The standard error of a proportion p is its standard deviation σ, which is in the case of random sampling:

$$Standard\ error = \sqrt{\frac{p(1-p)}{k}}$$

Based on this formula, Ojala and Garriga (2010) show how to determine the number of samples required to get a result with an standard error below a certain value:

$$\lim_{p \to \frac{1}{2}} \sqrt{p - p^2} = \sqrt{\frac{1}{4}} \Rightarrow Upper\ bound = \frac{1}{2\sqrt{k}}$$

The idea of significance testing is rather old but became more popular in recent years because computers make their application even on larger result sets possible (Noreen, 1989). A sketch of the algorithm we use for significance testing is given in Riezler and Maxwell (2005) in the context of statistical machine translation (SMT) (see Algorithm 2). This algorithm can be applied to basically every case where the results of a system is represented as a list of tuples of topics (e.g. queries and corresponding scores).

[12]This is a one-sided p-value. If the difference is calculated on absolute values, the result is a two-sided p-value.

Algorithm 2 Approximate Randomization Test for Statistical Significance Testing. From: Riezler and Maxwell (2005).

$c \leftarrow 0$
Compute actual statistic of score differences $|S_X - S_Y|$ on test data
for random shuffles $r = 0, \ldots, R$ **do**
    for sentences in test set **do**
        Shuffle variable tuples between system X and Y with probability 0.5
    end for
    Compute pseudo-statistic $|S_X^r - S_Y^r|$ on shuffled data
    if $|S_X^r - S_Y^r| \geq |S_X - S_Y|$ **then**
        $c \leftarrow c + 1$
    end if
end for
$p \leftarrow (c + 1)/(R + 1)$
Reject null hypothesis if p is less than or equal to specified rejection level.

Exactly the same approach was examined by (Smucker et al., 2007) for significance testing of information retrieval metrics. The authors provide a script for significance testing based on the output of the `trec_eval`-script mentioned before, so we gratefully use this implementation[13] in our evaluation.

A final note on our evaluation settings regarding the significance level: unless otherwise noted, we use a significance level of $p = 0.01$ to estimate statistical significance of a system change when we compare models or systems.

[13] `http://www.mansci.uwaterloo.ca/~msmucker/software.html`, accessed 06.08.2013

Chapter 4

Monolingual Information Retrieval

In this chapter we examine the applicability of our model to monolingual information retrieval. We start in Section 4.1 with the description of the steps needed for data creation and explain the task of monolingual information *retrieval on the 20 NG corpus*. After preliminary considerations to motivate our future strategy, we start in Section 4.2 conducting *experiments* aimed at our main concern, feature selection and reduction. We end this chapter in Section 4.3 giving a roundup of the *results* of the experiments, hoping to transfer some of the findings to the upcoming experiments.

4.1 Retrieval on the 20NG Corpus

The models we review in this chapter are variations of the basic model (Equation 2.13). During the experiments, we use cosine similarity on normalized TFIDF vectors as baseline and evaluate the following two variants of the model:

- learning the diagonal
- learning the full matrix

The results of the three systems trained under different settings are then compared in the experiment's Section 4.2.

Data

For the monolingual experiments throughout this chapter, we use the 20newsgroup (20NG) corpus [1]. This almost historic dataset is frequently used in IR tasks and there exist baselines we can compare our results to.

The original 20NG dataset consists of approximately $20,000$ posts taken from 20 newsgroups (see Table 4.1). The archive we used for data set construction is the "bydate" version,

[1] `http://qwone.com/~jason/20Newsgroups/`, accessed 23.08.2013

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Table 4.1: Overview of contained groups of the 20 newsgroup corpus as they are grouped on the original web page.

which contains posts sorted by date and is already split into training (60%) and test (40%) sets. Duplicates and newsgroup identifying headers were removed from the posts, leaving in total $18,846$ documents.

We apply the following steps for preprocessing the documents: 1) All words are lowercased. 2) All numbers are removed. 3) Punctuation symbols are also removed. 4) the documents are finally cleaned (e.g. multiple whitespaces removed).

Some statistics on the corpus after these preprocessing steps are listed in Table 4.2. One important observation is that although the numbers of posts are relatively even among newsgroups (between 999 and 628 posts), the lengths of posts depends on the newsgroup. For example, posts in group *misc.forsale* have an average length of 142.35 words, posts in group *talk.politics.mideast* contain more than three times more words on average, i.e. 509.6 words.

Task Description

The 20NG corpus is a standard dataset for many text applications of machine learning tasks. It is mostly used for text classification and text clustering, however, it can also be applied to ranking and other information retrieval tasks (Joachims, 1997).

In our setup, we evaluate the information retrieval task on the 20NG corpus as defined in Chen et al. (2010): given a post of a newsgroup from Table 4.1, the task is to rank all posts of this newsgroup higher than all posts of all other newsgroups.

4.2 Experiments

Our baseline, ranking by TFIDF scores, is actually very simple to calculate: the vectors are normalized to unit length, so we just need to apply the basic model using the identity matrix $\mathbb{I}_N$, i.e. a $N \times N$ matrix with the diagonal constrained to 1 and all other values set to 0. This is already a relatively good baseline, because TFIDF is able to filter function words and gives weight to content words. The latter words can sometimes be precise indicators for

	# posts	∅ length
training documents	11,314	315.20
test documents	7,532	291.64
alt.atheism	799	357.32
comp.graphics	973	271.81
comp.os.ms-windows.misc	985	499.66
comp.sys.ibm.pc.hardware	982	206.65
comp.sys.mac.hardware	963	192.45
comp.windows.x	988	319.13
misc.forsale	975	**142.35**
rec.autos	990	230.00
rec.motorcycles	996	209.15
rec.sport.baseball	994	233.28
rec.sport.hockey	**999**	262.09
sci.crypt	991	343.69
sci.electronics	984	214.61
sci.med	990	305.67
sci.space	987	298.04
soc.religion.christian	997	393.88
talk.politics.guns	910	358.20
talk.politics.mideast	940	**509.64**
talk.politics.misc	775	451.37
talk.religion.misc	**628**	374.07
∅ per group	942.3	304.77

Table 4.2: Training and test splits and per group statistics for the 20NG corpus. Minimum and maximum values are printed in boldface.

newsgroups: for example, the word "atheism" is likely to occur in a discussion within the *alt.atheism newsgroup* and will rarely be observed in posts of other newsgroups.

Learning the diagonal does a re-weighting of terms in accordance to the IR task. This approach is very useful in information retrieval, because the weight of a term (e.g. TFIDF or word occurrence) is adjusted according to a certain information need. This results in higher weights for words that are more important for newsgroup affiliation. This way learning the diagonal adapts to the requirements of the task and will refine the initial TFIDF values.

The full matrix as the most powerful model is able to capture all possible relationships between words. Each query word is paired with every document word to capture dependencies such as synonymy, polysemy and antonymy. The models high capacity entails the well-known problems of training complexity and especially overfitting. Thus, generalization capabilities and overfitting are issues that have to be kept in mind.

Convergence

A central question for SGD-based learning algorithms is how many training instances should be processed until the model starts to degenerate. In the machine learning framework, the perfect number is reached when the model performs best on unseen data.

The usual approach is to evaluate the model after n steps on development or held-out data and calculate an error metric. As soon as the error metric stops falling and starts rising, learning should be stopped, because this is what overfitting actually means.

Evaluating each model after n steps is a time-consuming task. Furthermore, the number of examples required to reach the optimal point depends on multiple factors involving model parameters, e.g. learning rate, feature representation, e.g. number of dimensions, kernels used etc. We want to estimate a good example count, but we don't want to do this for every change of settings.

As it is too costly to do this evaluation for every model and parameter combination, we first examined several standard cases to get an idea how many examples are needed and at which settings the model is in danger of degenerating. Based on these observations, we determine a number of training instances that represent a compromise between learning complexity and task performance.

Progressive validation (Blum et al., 1999), a built-in feature of the VW learning tool we use, showed up to be extremely helpful for getting error metrics during training. The basic idea of progressive validation is to simulate hold-out estimates by incrementally testing on unseen training data before it is actually send to the learning algorithm. The neat trick is to create multiple hypothesis with corresponding error outputs during training and then *sample* from the evaluates to get an estimate. The expected number of unseen examples used by any hypothesis is then $\frac{m-1}{2}$ and the authors show that by preserving the same guarantee as the holdout bound it is a good estimate of the generalization error.

The development of the progressive error during training on the 20NG dataset is displayed in figures 4.1 and 4.2. For the diagonal model the error decreases very fast, while for the quadratic model this takes considerably longer to reach the point where the error converges.

The progressive error and the empirical MAP scores on the test set plotted in Figure 4.3 show a high level of negative correlation to the progressive error. Judging from this result, we conclude that 2 million pairs are a good starting point to get reliable results on the test performance for the comparison of models and parameters.

Fixed vs. Decaying Learning Rate

As a first experiment, we compared fixed learning rate (FLR) against decaying learning rate (DLR). The formula for the learning rate η_t used by the machine learning tool is:

$$\eta_t = \lambda d^k \left(\frac{t_0}{t_0 + w_t} \right)^p \tag{4.1}$$

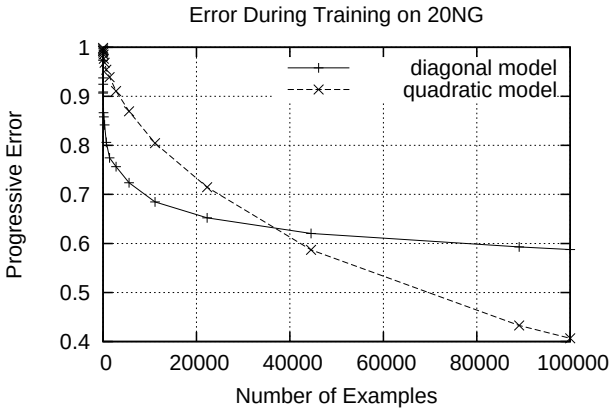

Figure 4.1: Error at beginning of training.

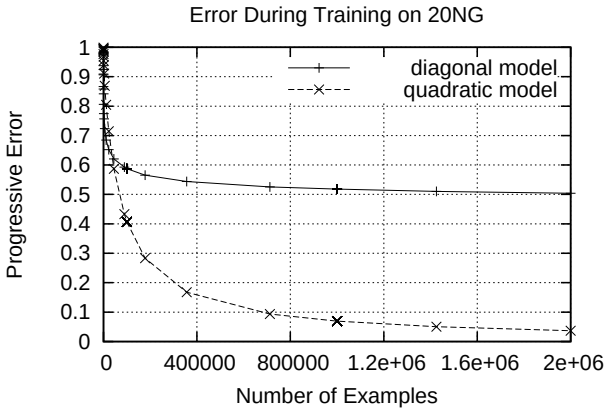

Figure 4.2: Convergence of the error.

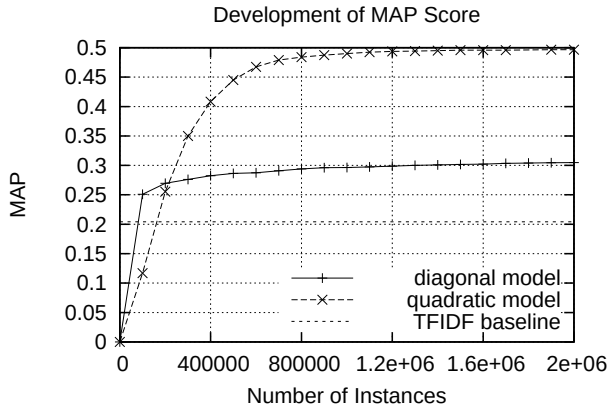

Figure 4.3: MAP scores on test during training on 20NG.

The parameter t_0 controls the influence of the sum of importance weights $w_t = \sum_{t'<t} i_{t'}$. Usually these weights are set to 1 and the value w_t represents the number of examples seen so far. With the default value $p = 0.5$, the square root of the term in parenthesis is taken. This has the effect of a moderately decreasing learning rate that converges to 0 very slowly. Setting the parameters $p = 0$ and $k = 0$ actually sets a fixed learning rate that is equal to the λ parameter.

Again, we looked at the progressive error (Figure 4.4) to estimate the number of required training instances and evaluated the models on the test set. The predicted error for the quadratic model seems to be wrong[2] and does not correlate well to the MAP scores in Figure 4.5. We also tried setting the learning rate to $\lambda = 200$, a value that Chen et al. (2010) used in their experiments, but for us the results changed only marginally.

Figure 4.5 plots the MAP scores against the number of training instances. All four models

[2]This is actually the only time where the learning tool we use does not work as expected.

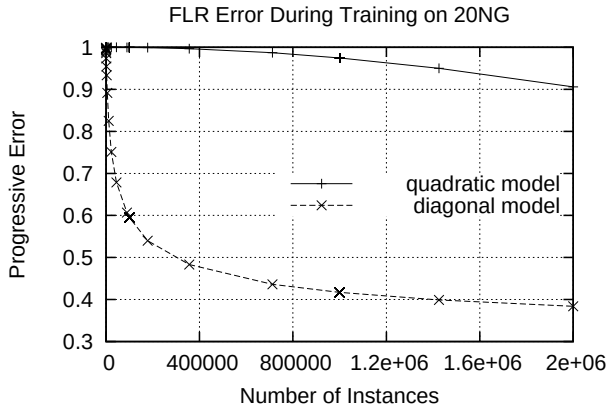

Figure 4.4: Error on diagonal model for fixed learning rate $\lambda = 0.5$.

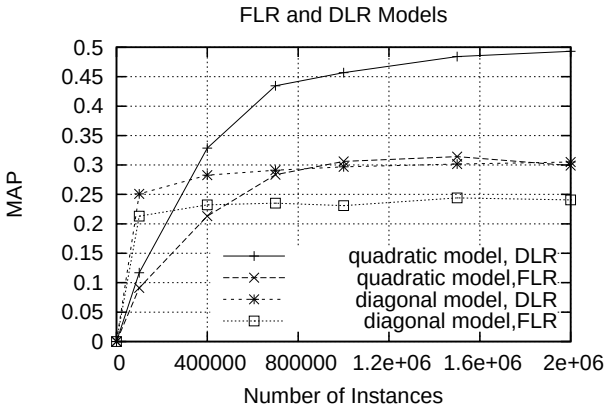

Figure 4.5: MAP scores on test set for DLR and FLR learning modes.

beat the TFIDF-baseline of 0.2041 MAP after seeing 100,000 instances. Although there are only few data points, the MAP-curves for FLR and DLR variants of the same model have a similar shape with DLR achieving significantly higher scores. For the exact scores after 2 million training instances, see Table 4.3.

	no train	DLR	FLR
identity	0.2041	–	–
diagonal	–	**0.3050**	0.2406
quadratic	–	**0.4948**	0.4660

Table 4.3: Comparison of MAP scores for fixed and decaying learning rates ($\lambda = 0.5$) after 2 million examples.

To be sure that we had not accidentally selected a bad learning rate, we searched logarithmically for useful learning rates in the interval from 10^{-3} to 10^{3}. The final result remains unchanged: Table 4.4 illustrates that MAP score after processing 2 million instances does not increase in any case.

learning rate	10^{-2}	10^{-1}	10^{0}	10^{1}	10^{2}	10^{3}	10^{4}
diagonal	0.2818	0.2574	0.2393	0.2353	0.2289	0.2264	0.2347
quadratic	0.2686	0.2976	0.2350	0.1116	0.1096	0.1096	0.1096

Table 4.4: MAP after 2 million examples and different fixed learning rates.

For SGD, a fixed learning rate generally complicates convergence, because every stochastic update gets the same weight. Thus, it can happen that the algorithm "jumps around" the optimal solution without ever reaching it. We conclude that in our setting a dynamic

learning rate works considerably better than a fixed one, although other works successfully applied a fixed learning rate to similar tasks using comparable models (Bai et al. 2010, Chen et al. 2010).

Feature Selection

In this section, we start our experiments on methods that actively select useful features. The first method is an embedded wrapper based on ℓ_1-regularization (see Section 2.2). The second selection method is actually an accidental discovery. There are various interpretations, but from one perspective, it is a filter method, comparable to reducing the dictionary in a more clever way that just applying a cutoff.

L1-Regularization

The idea of regularization is to increase generalization capabilities by adjusting the parameters of the model with a certain strategy. From the optimization problem's perspective, this is accomplished by adding a regularization term to the objective function (see optimization problem 2.7). This term usually involves a norm of the model parameters, which is in our case the weight vector.

To enforce sparsity, we apply ℓ_1-regularization known as LASSO (see Section 2.2). The ℓ_1-regularizer term in the optimization problem 2.10 effectively pushes the values to zero in each step while each gradient update increases them. Regularization is useful because it helps against overfitting and to get a better generalizing model. However, finding the right regularizer is expensive as the space of possible regularization settings usually has to be explicitly searched.

For our retrieval task on the 20NG corpus, we tried several values for ℓ_1-regularization. The MAP scores against the number of training instances are plotted in Figure 4.6, illustrating the effect of the regularization on MAP and convergence.

The graphs for λ values between 0 (regularization off) and 10^{-8} have very similar shapes. As soon as the regularization term has enough weight to actively influence the model, the MAP score drops at a certain point and overfitting occurs. This can be seen clearly in the graph for $\lambda = 10^{-6}$, but it is also present for $\lambda = 10^{-7}$.

Table 2 summarizes the best achieved MAP scores and adds the corresponding model sizes. The bottom line for our experiment on 20NG is: as soon as the regularization is able to decrease model complexity, performance drops. The values indicate that enforcing sparsity with ℓ_1-regularization does not help in this case.

We did not test the influence of regularization on the diagonal model, because we could not think of any positive effect for the diagonal case. Either the values are pushed to 0 (ℓ_1-regularization) effectively removing weights, or the values are raised (ℓ_2-regularization), meaning that the model converges to the standard TFIDF model.

Our interpretation of the experimental results on ℓ_1-regularization is that the underlying task of identifying documents from the same newsgroup is not prone to overfitting because

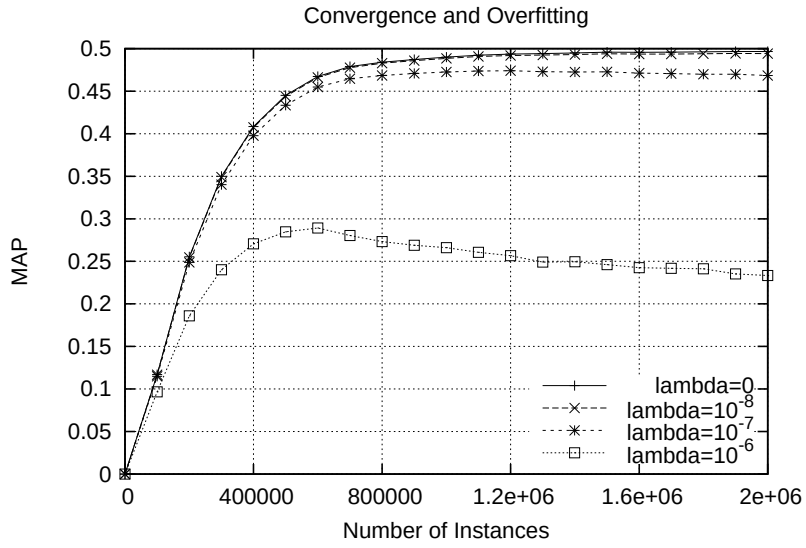

Figure 4.6: Influence of ℓ_1-regularization on convergence and overfitting.

	10^{-6}	10^{-7}	10^{-8}	10^{-9}	0
best MAP	0.2892	0.4740	0.4943	0.4964	0.4966
(model size)	96M	759M	1038M	1068M	1073M

Table 4.5: Best MAP score achieved under different ℓ_1-regularization settings and corresponding model sizes.

there are many "correct" or "relevant" documents (a newsgroup has an average size of 942.3 posts) which include a lot of noise, so the danger of adapting "too close" to the training data is small.

TFIDF Weighting Scheme

In this experiment we examined the influence of the TFIDF-weighting on the learning quality. The DF-values, i.e. the document frequencies for words, are often calculated on a completely separate collection of documents. For reasons already mentioned before, finding an appropriate document collection with similar properties is difficult, and the existing document set is too small to be further divided. So, we were forced to use only the existing 20NG data but applied three different counting schemes for calculating the DF-values needed for creating the TFIDF vectors:

1. For training use DF-values from training set, for testing combine document frequencies from training and test set.

2. For training and testing, use DF-values from the training set only.

3. For training and testing, use DF-values from combined training and test set.

The best performing counting scheme of document frequencies is variant 2, as the results in Table 4.6 clearly show. Although the differences to the other variants for identity and the learned diagonal are not significant, for the quadratic model this variant performs considerably better and exceeds the results reported by Chen et al. (2010) by a large margin (Chen's best system scores MAP 0.426).

	train/train+test (variant 1)	train only (variant 2)	all (variant 3)
identity	0.2041	**0.2055**	0.2041
diagonal	0.3041	**0.3050**	0.3045
quadratic	0.4660	**0.4948**	0.4632

Table 4.6: MAP score for different TFIDF construction schemes.

Our first interpretation for the lower scores of the first variant was that it uses different DF-values and thus different word-document statistics for training and test set. This is clearly a violation of the basic assumption for supervised machine learning, because training and test set must be i.i.d., otherwise the learning performance is spoiled. Judging from the last variant, this explanation cannot hold. Variant 3 was initially thought to give an upper bound of achievable scores and we never planned to use this scheme for the experiments, because it is prohibitive to transfer any statistical information from the test to the training set.

The TFIDF construction we finally apply is variant 2: this scheme basically removes all terms that are not part of the training set. The fact that both training and test must be sampled from the same distribution is respected, but there is another aspect regarding hash map collisions because of the hashing trick we apply to the parameter vector (Section 2.5). Words not seen during training but encountered during test are either hashed to an empty bucket or to an existing bucket. Hashing to an empty bucket will return a 0 weight and discards the feature, practically doing no harm. Hashing to a non-empty bucket, however, is an unwanted collision and will contribute to the information loss. Thus, removing those features will never harm but it helps to reduce unwanted collisions.

We see variant 2 of the TFIDF construction as a selection method that removes rare words for which no sufficient statistics exist. By doing so, this construction includes a very rudimentary filter method for feature selection.

Reducing Feature Space

Reducing the feature space is a filter method that aims at reducing model complexity. However, in the case of the learning tool we use, the parameter space is fixed and weights are hashed into this space. Determining model complexity based on the size of the model alone is thus inaccurate.

In this section, we examine two approaches to actively reduce the feature space of our basic model by limiting the number of distinct words. The second approach is based on hash kernels which we successfully apply to decrease information loss by increasing feature redundancy.

Limited Vocabulary

The original vocabulary contains 115,141 words with 45,488 words occurring only once, many of them being wrongly spelled words. These singletons are a problem for any learning task, because word pairs build with them are rare and occur only in the training or in the test set (or in neither). These pairs do not contribute positively to the learning task but they merely increase the number of collisions.

An obvious way to reduce collisions is to limit the dictionary size. The most simple approach applies a cutoff at a certain position in the word frequency table and keeps only the top-k most frequent words. We limited the dictionary to 10,000 words, which means filtering everything besides the top 10,000 most frequent words.

The decision to use 10,000 words as vocabulary limit is not completely arbitrary. One reason is that we want our results to be comparable to Chen et al. (2010). Another reason is the reduced complexity for the model: on 10,000 words, we can generate a maximum of 100,000,000 different word pairs which is much less than $115,141^2$ on the original vocabulary size.

With a perfect hash function, a parameter vector of size 10^8 would be sufficient to have no collisions. Perfect hashing is just a theoretical idea here but a realistic estimate is given by birthday paradox: with n buckets in the table the probability of getting a collision exceeds 0.5 for $m \approx \sqrt{2m \cdot 0.5} = \sqrt{m}$. This means, with $\sqrt{10^8} = 10,000$ entries the probability of getting a collision is already at 0.5.

However, it is difficult to lower these bounds and not every collision results in information loss that harms the learning task. As our learning tool uses a parameter vector of size 2^b, we set $b = 27$ for the quadratic model to get a vector of size $2^{27} = 134,217,728$. In the optimal case, this parameter vector uses $2^{27} \times \text{sizeof(float)} = 536,870,912$ bytes. For the diagonal model, we set $b = 18$, i.e. 262,144 entries in the weight vector, which makes collisions not completely unlikely but rare.

The current quadratic model requires about 2 GB for the model in memory and 1 GB for the model on disk, which are very moderate hardware requirements making it possible to train multiple models in parallel, e.g. to search for optimal configuration parameters.

	full	limited	relative change
identity	0.2041	0.2054	1.006
diagonal	0.3050	0.2826	0.927
quadratic	**0.4948**	0.4536	0.917

Table 4.7: Influence of limited and full vocabulary on MAP for several models.

Word	Mappings				
epidemic	providers	accine	infected	genes	killer
liklihood	contention	acknowledge	likelihood	implying	difficulties
chemotherapy	radiation	treatments	occuring	histories	lone
painless	cruel	viewers	painful	execution	gifs
tinnitus	ears	iss	loud	depression	sleep

Table 4.8: Sample mappings created by correlated feature hashing on 20NG.

An interesting question concerning the limited vocabulary is if the information loss by discarding words is lower than the loss by hash collisions. Table 4.7 lists MAP scores after training different models on 2 million examples with limited and full vocabulary. For the retrieval performance of the identity, the limiting makes no difference, because TFIDF is already a relatively poor strategy for the retrieval task.

The rows for the diagonal and the quadratic model in Table 4.7 support the assumption that limiting the dictionary destroys more information than the collisions because of hashing. In fact, collisions seem to have less influence than expected: for the diagonal model, there are theoretically $262,144/10,000 \approx 26.14$ buckets per feature, for the quadratic model only $134,217,728/100,000,000 \approx 1.34$. In both cases, the relative change in scores is comparable, suggesting that the increased collisions do not spoil the model automatically.

Correlated Feature Hashing

The method named *correlated feature hashing* explained in Section 2.5 is another way to effectively limit the vocabulary and thus reduce model complexity. The disadvantage of using a simple cutoff based on vocabulary size is that information is destroyed as seen in the previous experiment. In addition to this, it is a reasonable argument that for information retrieval tasks the rare words are especially important, and these are the ones discarded while limiting. On the other hand, a standard vector space model evaluates document matches in vector space, so it is difficult to gain information from rare words and impossible to learn something useful from singletons.

By applying correlated feature hashing (CFH), rare words are mapped to co-occurring frequent words. This method is more sensitive to contained information than a simple cutoff procedure. CFH concurrently increases learning coverage, because mid-frequent words can contribute more to the learning objective than rare words. In addition to this, multiple binning reduces the consequences of collisions as explained in Section 2.4. For the 20NG corpus, we use CFH with 5 bins ($k = 5$, see Equation 2.24) and a base vocabulary size of $\mathcal{F} = 10,000$ words to get comparable results.

Table 4.8 contains some examples terms with their five most correlated words among the limited vocabulary sorted by DICE-score. Some mappings make sense, e.g. “epidemic” and “chemotherapy” are mapped to words that mostly have an clear semantic relation. It is interesting to see that CFH is able to select the correct form of a wrongly spelled

word: for "liklihood" the correction "likelihood" is among the top correlated words. Other semantic relations like antonyms are also recognized, for example the algorithm correctly selects "painful" as a correlated word for "painless".

However, due to the size and the nature of the corpus, there are several problems. A word that is often wrongly spelled like "occuring" in the third row can become part of the limited dictionary and will then be a valid target word for the mappings. In the 20NG corpus, the wrongly spelled word "occuring" is contained 25 times, which explains the preference for this string. Sometimes co-occurring words seem to be randomly selected because there exists no obvious connection between the original word and the mapping. However, this problem is less critical than it first seems: such words count or discount the mapped weight of a source word, but in the long run only words that have a real semantic relation will push the weight into the right direction. Words occurring too frequently in arbitrary mappings will not contribute to the learning and their weight will eventually converge to zero.

Ideally, the CFH mappings are calculated on a separate data set. For the 20NG corpus, this is difficult to accomplish, because the language in use is very special. On one hand, the corpus contains many specialist's terms of completely different subjects, e.g. atheism and computer hardware. On the other hand, a lot of spelling errors and colloquial terms are naturally contained in these discussion posts.

A mapping constructed on an external resource such as Wikipedia would therefore destroy a lot of information when applied to the 20NG corpus, because all colloquial and misspelled words will not get mapped. Thus, we decided to use 20NG for the mapping, but in order not to accidentally transfer information from the training to the test data, we generated the mapping for the training set using the training data only. Then, we compared the performance on two variants of creating the CFH mapping for the test set:

1. Mapping for test data based on the whole corpus (training and test).

2. Mapping from training data only, effectively dropping all rare words not present in the training data.

The motivation for the first variant is that we want to use as many information as possible. As it is forbidden to transfer knowledge of word distributions from test to training, we use the whole set only for testing. The second variant was first planned as a comparison how well the first variant performs. The whole setup is comparable to the experiment we conducted on the TFIDF counting schemes (see Section 4.2), and the result in fact revealed similarities.

Although the second variant apparently removes more information, the comparison of rows "CFH (full)" (corresponding to variant 1) and "CFH (train)" (variant 2) in Table 4.9 illustrates that the model evaluated against the test data constructed with the training-only mapping performs significantly better than if evaluated on test data constructed with a mapping covering the full vocabulary.

The explanation is again related to the statistics hidden in the data: the task in machine learning is to learn patterns, to reveal underlying structures by analyzing the distribution of

data. It is almost certain that we add some information in the first mapping variant that uses the full corpus, however, this goes together with a change of distribution of words. Patterns learned on the training set do then appear only partially in the test set, and new patterns hidden in the test set were never seen before. Thus, the score *must* go down, anything else would mean we made an error.

	CFH (full)	CFH (train)	CFH-1	full dict.
identity	0.2525	**0.2619**	0.1930	0.2041
diagonal	0.3795	**0.4136**	0.3000	0.3050
quadratic	0.5012	**0.5289**	0.3915	0.4948

Table 4.9: Comparison of MAP scores on different correlated feature mappings.

The third CFH-based value in Table 4.9 labeled "CFH-1" is a mapping where each word is mapped only to the top-most correlated word. This performs much worse than the other mappings and while it is on par with the diagonal on the full dictionary, it performs much worse on the quadratic model. Apparently, the restriction to map each word only to exactly one frequent word destroys too much information by discarding useful patterns in the data.

4.3 Results

The monolingual information retrieval experiments conducted on the 20NG corpus show that the basic system performs very well in the IR task examined in this chapter. The model is able to learn useful weights for word pairs as the comparison between identity and learned diagonal reveals. The quadratic model further improves IR performance by capturing more complex word relations such as synonymy, polysemy and antonymy.

Using hashing for the parameter vector did not show negative side effects in the experiments. The hashing trick enables the model to use the full vocabulary which significantly increases performance. The impact of the increased number of collisions is moderate compared to the information gain.

A surprisingly effective way to reduce unwanted collisions is to remove words from test data that were not observed during training. This is accomplished by using the DF-values of the training set for calculating TFIDF vectors for the test set. This efficiently reduces information loss as a consequence of collisions introduced by hashing.

We compare our work to the work of Chen et al. (2010), as they use a similar model but follow a different strategy based on enforcing sparsity. Their results on fixed and decaying learning rates are in accordance to ours: fixing the learning rate gives a significantly lower score than using a decaying learning rate.

However, most of the MAP scores we obtained differ from those reported by Chen et al. (2010). Using TFIDF scoring, our model achieves a MAP score of 0.204 compared to 0.185, and for the learned diagonal we obtain 0.304 compared to 0.190. Apparently the authors work on a slightly different dataset, as the difference for in TFIDF scores suggests.

	identity	diagonal	quadratic
FLR	–	0.2406	0.4660
limited	0.2054	0.2826	0.4536
full, $\lambda = 0$, TFIDF v1	0.2041	0.3041	0.4660
TFIDF v2 (DF-train)	†0.2055	†0.3050	†0.4948
$\lambda = 10^{-6}$	–	–	0.2892
$\lambda = 10^{-7}$	–	–	0.4740
$\lambda = 10^{-8}$	–	–	0.4943
CFH ($k = 5$, full)	0.2525	0.3795	0.5012
CFH ($k = 5$, train)	†**0.2619**	†**0.4136**	†**0.5289**

Table 4.10: Overview of MAP scores under different configurations. Values preceded by (†) indicate a significant change to the system in the previous row.

A way to intelligently reduce parameter space is to apply correlated feature hashing. This reduces the dictionary in such that collisions become rare while the semantic content is not spoiled. On a small corpus like 20NG, CFH is not always able to capture words with related meanings, but still seems to preserve useful patterns in the data. With a MAP of 0.5289, our final model achieves more than 10 MAP points more as the best model (Chen et al. 2010, Sparse-R, MAP 0.426).

However, we were not able to successfully enforce sparsity by using ℓ_1-regularization. Applying ℓ_1-regularization for 20NG always decreases MAP score. In our experiments, the regularization decreases MAP as soon as it affects the size of the model. Apparently, ℓ_1-regularization fails to select the useful features for the task. Although the nature of discussion posts brings along a lot of noise, the model is able to find useful word correlations without regularization.

Summarizing the experiments in this chapter, both diagonal and quadratic model showed strong performance compared to the TFIDF baseline and to other works. Besides the positive influence of CFH, the most interesting results is that the TFIDF construction scheme as well as the CFH mapping affect the scores significantly if DF-counts respectively mappings are calculated on the training data only.

Chapter 5

Cross-Language Information Retrieval

In this chapter, we examine how the basic models performs in a specialized cross-language information retrieval task and describe a method how to combine different models in a novel way. We start in Section 5.1 with the description of the task of *bilingual patent retrieval* and present the data we use for training and testing. In Section 5.2 we conduct several experiments on the sparse model to carefully select the best methods and optimal parameters for our final model. This section is concluded by a short overview on the intermediate results. We then present in Section 5.3 a novel method of *combining sparse and dense models*, where we take the best model from the previous section and combine it with highly informative domain knowledge features defined on the patent domain. At the end of the section, we demonstrate the flexibility of our approach by integrating a completely different model from another work. This chapter ends with Section 5.3, where we provide a conclusion on the final results of the experiments on the sparse and the dense model.

5.1 Bilingual Patent Retrieval

The basic model introduced in chapter 2 performed remarkable well on the monolingual IR task. Its advantage over standard retrieval models such as TFIDF or BM25 is that it is able to learn not only a useful re-weighting of matching terms (as for the diagonal model in Section 4.2), but also to consider all possible word pairs and weigh them according to the given task.

This quadratic model is able to capture different word relations that are not restricted to the diagonal of the matrix, where only direct word matches count. A logical step is to apply the quadratic model on word pairs that are not from the same language. Here, standard TFIDF score and the diagonal model cannot be applied in a reasonable way, but the quadratic model has the potential to develop a useful similarity or relevance measure to retrieve information from a collection written in a language other than the query language.

Data

The data we used for training and testing the cross-language information retrieval (CLIR) model is based on patents. There are several reasons for this, most notably, patents have a clear structure and are written in a very precise language.

Fortunately, we were able to use data created by our research group especially for CLIR on patents. This corpus named BoostCLIR[1] is licensed under a *Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License*[2] and was created during the works of Sokolov et al. (2013).

The data was constructed following a method described by Graf and Azzopardi (2008), where relevance judgments are extracted based on the citations of a patent: a patent p that is cited by another patent q is considered relevant to q. If the citation was added by an examiner, it is considered higher relevant than if the citation was added by the applicant. A special case is if patent p and q are in the same family, i.e. the patents are related to the same invention but granted by different authorities. Then, patent p is considered highly relevant for patent q. The relevance matrix based on these rules is:

$$\mathbf{M}_{ij} = \begin{cases} 3 & \text{if patent } p_i \text{ and } p_j \text{ belong to the same family} \\ 2 & \text{if patent } p_i \text{ cites } p_j \text{ by the examiner} \\ 1 & \text{if patent } p_i \text{ cites } p_j \text{ by the applicant} \\ 0 & \text{otherwise} \end{cases} \tag{5.1}$$

A similar construction algorithm was suggested by Guo and Gomes (2009), but the authors did not implement the highest rank based on family relation, because they evaluated a monolingual patent task.

The relevance judgments were extracted from the MAREC[3] corpus, the document data from a combination of MAREC and the NTCIR[4] patent data. As MAREC contains only English abstracts, the corresponding Japanese abstracts were extracted from the NTCIR corpus to build the final resource with Japanese query patents and English document patents. Table 5.1 contains an overview on the document sizes, relevant links between documents based on the matrix (5.1), the average number of terms in patent abstracts and the sizes of the English and Japanese vocabulary, $\mathcal{D}_{en}$ and $\mathcal{D}_{jp}$.

The final corpus we use contains only Japanese and English abstracts and document IDs. Since we use the bag-of-words model to learn the CLIR model, we consider short documents as beneficial for our method. The highly specialized vocabulary of patents is another positive aspect that makes correlations between certain terms more useful than for other data sets.

Finally, for the experiments in Section 5.3 involving domain knowledge features, we extracted additional meta-information using a script originally by Ruppert (2013) which we

[1] http://www.cl.uni-heidelberg.de/statnlpgroup/boostclir/

[2] http://creativecommons.org/licenses/by-nc-sa/3.0/

[3] http://www.ifs.tuwien.ac.at/imp/marec.shtml

[4] http://research.nii.ac.jp/ntcir/data/data-en.html

	all	**train**	**dev**	**test**
# docs_{en}	1,088,127	888,127	100,000	100,000
# docs_{jp}	111,061	107,061	2,000	2,000
# rel. links	1,473,904	1,422,253	25,173	26,478
∅ len_{en}	135.63	136.98	128.79	128.48
∅ len_{jp}	154.43	154.34	157.00	156.97
$\mathcal{D}_{en}$	365,553	305,257	110,637	111,214
$\mathcal{D}_{jp}$	60,470	59,284	10,014	9,834

Table 5.1: Characteristics of the Japanese-English patent corpus. The value ∅ $\text{len}_{\{en,jp\}}$ denotes the average number of words per abstract, $|\mathcal{D}_{\{en,jp\}}|$ is the number of distinct words.

modified to our needs. The meta-information is read by our system and the dense features are created on-line by querying a database.

Task Description

For cross-language information retrieval experiments we examine the application of *prior art search*: given a patent application, the task is to retrieve a list of existing patents that are relevant, i.e. considered useful as prior art by applicants and examiners. It is important to extend this idea to the cross-language setup, because prior art disclosures are valid independently of the language used.

5.2 Experiments on the Sparse Model

The initial experiments are aimed at finding the number of training instances required to further evaluate the system. As in the previous chapter, we first observe how the error curve and the score function converge with the data. Then, we compare several preprocessing steps and their impact on the learning task. Feature selection and several filter methods for feature space reduction are in the focus of this section. At the end of this section we give a brief intermediate analysis on the results of the sparse model.

Preparation

In accordance to the monolingual experiments, we started with observing how the error curve behaves and how fast the function converges during training. This lets us estimate the number of training instances required to compare models under different settings.

Again, we plotted the progressive error curve (Blum et al., 1999) for the unregularized basic model. The TFIDF-vectors to generate the training data were built according to variant 2 in Section 4.2. For comparison, we also plotted binarized vectors, where all individual word weights are removed s.t. a word is either present (value 1.0) or absent (value 0).

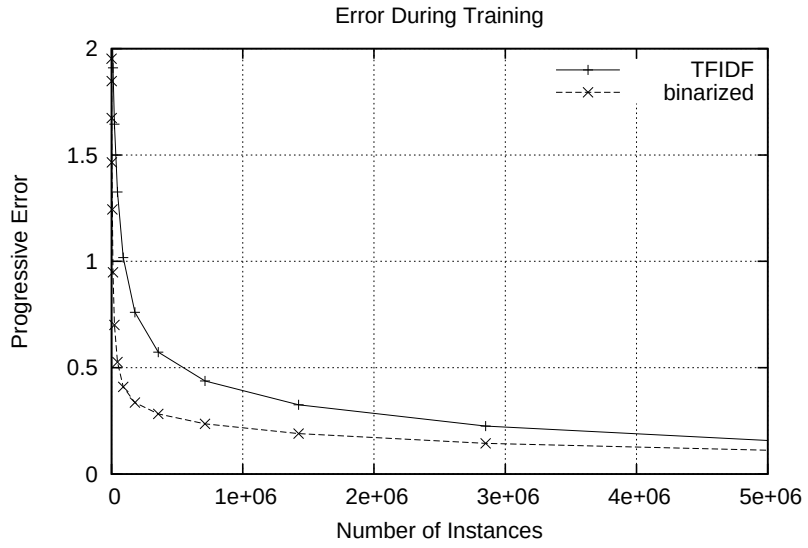

Figure 5.1: Progressive error during training on TFIDF and binarized data.

Figure 5.1 demonstrates that the error during learning on binarized data drops faster than for TFIDF. The explanation is that TFIDF enriches the vector representation so that the learning converges much slower. Judging from the error graph, we decided to use approximately 5 million training instances for the first experiments. After this amount of instances both models seem to perform reasonably well, although the TFIDF model will benefit from more training.

Vector Binarization

When we first applied the basic model to the cross-language task, the results were disappointing. The scores evaluated on the development set were very low, although we tried different training sizes and regularization parameters. In fact, the results were so bad, that we searched for errors in each step of the preprocesssing pipeline, the learning module, and the evaluation, but we could not find the source of the problem. Meant as a sanity check we finally learned on training data that was prepared for a completely different model based on boosting where the vectors were binarized – and the basic model started to work.

Figure 5.2 contains an interesting range of the extensive search for the right model parameters. The lower graph represents the scores after training with TFIDF vectors varying λ for ℓ_1-regularization between 0 and 10^{-6} and trained on 5 million instances. The changes of score are marginal and the area where the scores reside is not worth discussing. However, the upper graph depicts the same λ-region while training on binarized vectors. Here, the binarization gave a huge increase in MAP score by a factor of approximately 12 compared to the TFIDF variant.

In a final attempt to rescue the TFIDF idea we mixed TFIDF and binarized vectors.

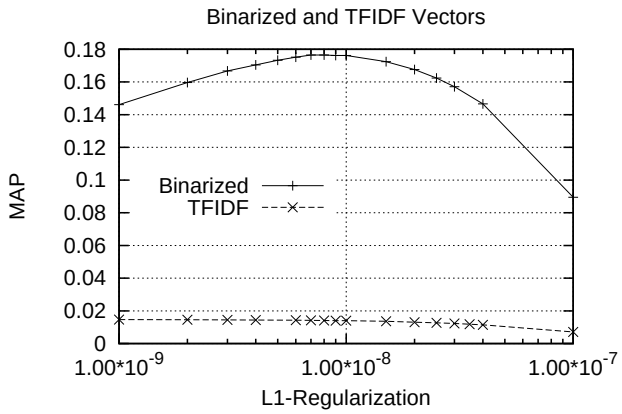

Figure 5.2: Comparison of learning on binarized vs. TFIDF vectors.

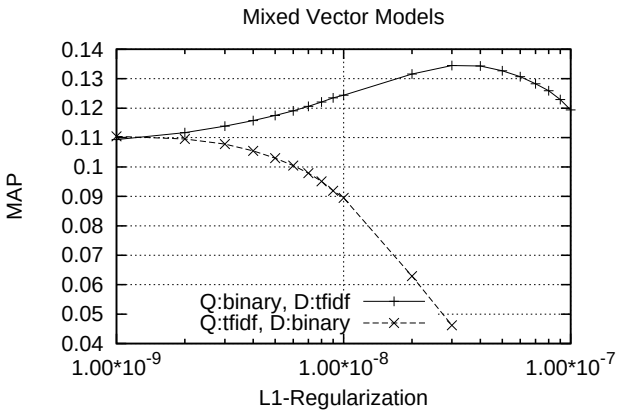

Figure 5.3: Test after training on mixed vector models.

The experiment was to use for both training and testing binarized vectors on one side, and TFIDF vectors on the other. The result graphs of MAP score against learning with different ℓ_1-regularization settings are plotted in Figure 5.3.

For reasonable low λ both systems achieve similar scores and perform much better than the "pure" TFIDF-approach. With increasing regularization the models' scores diverge with a remarkable higher score for the model that uses binarized Japanese and TFIDF English vectors. This might indicate that the problem with TFIDF scores comes mainly from the Japanese side. However, the model cannot beat the scores of the system using only binary vectors (Figure 5.2).

Judging from the results in Figure 5.3, we did a final experiment on the model with binary query- and TFIDF document-vectors, where we increased the number of training instances by running multiple epochs over the data. For the settings we checked, $\lambda = \{10^{-9}, 5 \cdot 10^{-9}, 10^{-8}, 5 \cdot 10^{-8}\}$ increasing training data does not help and the scores degrade after the first epoch. Thus, we decided not to put more effort into training on TFIDF weighted vectors at this point.

Our explanation for the disappointing performance of the TFIDF model is based on the fundamental differences between the two languages Japanese and English: as a consequence, TFIDF weights do not supply useful information for learning a similarity metric but merely add noise, making it more difficult for the learner to extract useful patterns. This assumption is supported by the observation that leaving one side binarized increases performance.

Binarization basically flattens the data and this makes it easier for the algorithm to select the pairs and extract the structures that contribute most to the learning objective.

Stop Word Removal

In the field of information retrieval, stop (or function) word removal is a standard method to simplify the model and remove noise. Function words are the grammatical "glue" between other words in sentences, but they carry practically no lexical information. As soon as they are taken out of the context and put into a bag-of-words as in our model, they become completely useless.

There exist various lists of stop words not only for languages but also for tasks, because it depends on the type of text which words can be removed without harming the retrieval performance. From the retrieval point of view, stop words are useless because they virtually occur in every sentence, thus returning a match for every document. TFIDF weighting exactly addresses this problem, because in those cases the inverse DF-component of the weighting becomes very small (DF is high), assigning such words very low weights.

english	a, an, and, by, for, in, is, of, the, to, with
japanese	が, こと, さ, し, する, た, て, で, と, に, の, は, れ, を
	図, 手段, 提供, 解決, 課題, 選択

Table 5.2: Words removed because their TFIDF weight is below 10^{-10}.

Instead of applying questionable stop word lists that may not work well for our purpose, we simply use the TFIDF weighting as a criterion for stop word removal. If the TFIDF-value is below 10^{-10}, we discard the term as its weight is to low to contribute in a useful way. Table 5.2 lists the words that we removed based on their weight.

The English side needs no explanation, they are unquestionable function words. For the Japanese words, the first line contains *hiragana* particles which are in fact true function words, such as の ("of") and は ("is"). The second line contains more complex words in *kanji* characters. These words still occur very often in this type of text, as the string 課題 means "abstract" in English, and 選択 図 are the "provided figures". In fact they represent the headings for corresponding sections and are present in every Japanese document of our corpus. This demonstrates that TFIDF-based stop word filtering is favorable because it can capture special properties of the underlying documents.

The filtered words are only few in numbers, but deleting them had an unexpectedly positive influence on the retrieval performance. The MAP scores went up by up to 10% as shown in Figure 5.4. We conclude that for our bag-of-words model based on unigrams stop words only add noise. However, higher order models like bi- or tri-grams are able to capture the information contained in these function words.

Finally, an important point to note is that the optimal regularization parameter λ depends on the preprocessing of words. As the curves in Figure 5.4 illustrate, the optimal λ is below 10^{-8} for the full vocabulary, while it is slightly above this value after stop word filtering.

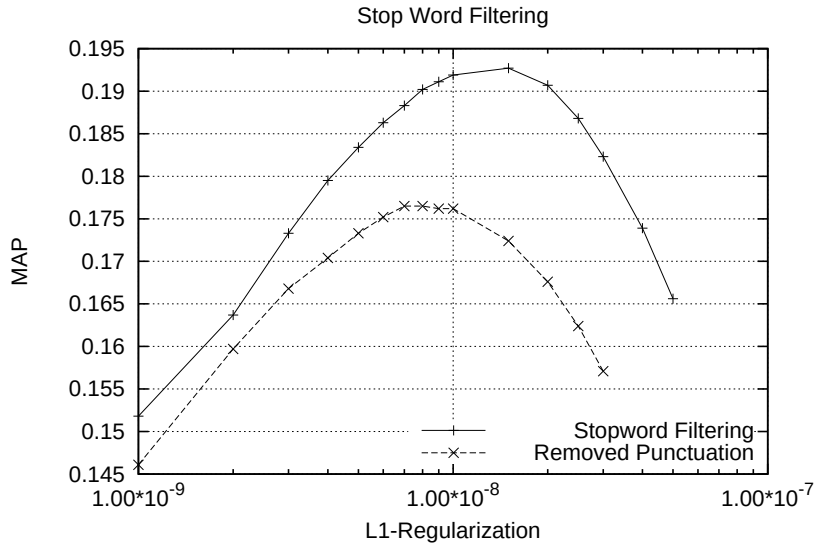

Figure 5.4: Effect of TFIDF induced stop word filtering on MAP score.

SGD Sampling Schemes

The 20NG training corpus from the previous chapter contains 20 newsgroups with 376 posts on average in each group. For pairwise training, the idea is to select a positive and a negative example, i.e. a post from the same and a post from a different newsgroup. So, for each "query document" on average $376 - 1 = 375$ positive and $7532 - 376 = 7156$ negative examples exist. The total number of possible training pairs is thus enormous and still very high if each positive example is to be considered at least once for training.

As for the patent corpus, the number of relevant documents per query is small (≈ 10 according to our experiments) and this makes it possible to think of sampling schemes for (q, d^+, d^-) other than selecting queries and documents completely independent:

1. Select a query q, sample a positive and a negative document (d^+, d^-) and create the training instance $(q, d^+ - d^-)$, then start over. This is what we usually do.

2. Select a query q, then sample n document pairs (d^+, d^-), each gives $(q, d^+ - d^-)$.

3. Select a query q, then iterate over all relevant documents d^+ and sample an irrelevant counterpart d^-, returning a sequence of $(q, d^+ - d^-)$ instances.

For the sake of completeness we name a fourth scheme, i.e. building all possible pairs (d^+, d^-) for a given query q. This is in most cases impractical: even on the small sized 20NG corpus this would mean to create over 2.8 million pairs *for each* query.

For our patent corpus, the ratio of positive and negative documents is completely different to the 20NG corpus. For a single patent, there are approximately 10 relevant patents on

average. Thus, it is not only possible to randomly sample query-document-pairs independently as in the 20NG experiments, but it is also possible to randomly draw a query and then create pairs involving all relevant documents.

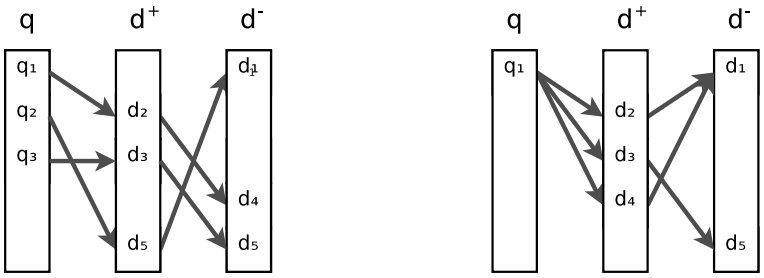

Figure 5.5: (left) For each of the three queries, a single random document-document pair is sampled. (right) The alternative scheme proposes to pair a single query with all relevant documents and random irrelevant documents. Here, both schemes generate three pairs $(q, d^+ - d^-)$.

Figure 5.5 illustrates the difference between the sampling schemes. The left figure corresponds to variant 1, and the right figure to the last scheme (variant 3). Variant 2 is a special case of variant 3 and means selecting a subset of n triples from the right figure.

Although differences should vanish in the long run, Table 5.6 shows a clear advantage for the alternative sampling scheme. One explanation is based on the "translating aspect" of the model: concentrating on one query and looking at 10 times more word pairs containing the current query words adds more useful information to the learning task than looking at the same number of more randomly chosen word pairs that have less structure and thus add more noise.

We applied the best regularization parameter so far, $\lambda = 10^{-8}$, and trained three models with training data created as described above. The files created according to schemes 1 and 2 contain exactly 5 million training instances, variant 3 has 5,057,980 training instances due to rounding errors. To see at which point overfitting starts, we let the algorithm pass 10 times over the data. The result is shown in Figure 5.6.

Right after training on approximately 5 million instances, the achieved scores are already quite different. While scheme 1 and 2 start at similar levels, variant 3 is a half MAP point above. This advance remains until all variants start to degrade because of overfitting. Even more interesting is the observation that variant 3 reaches the level between 4 and 5 epochs, while variant 2 needs 7 epochs to reach its peak.

The differences are not high, at least between variant 2 and 3, but there is a clear advantage for the new sampling schemes over the old one. Under optimal conditions (data size, regularization parameter, learning rate) all variants should lead to similar results. However, in our experiments the new scheme returns initially higher scores *and* converges faster, so from now on, we are using the new sampling scheme to create the SGD training instances.

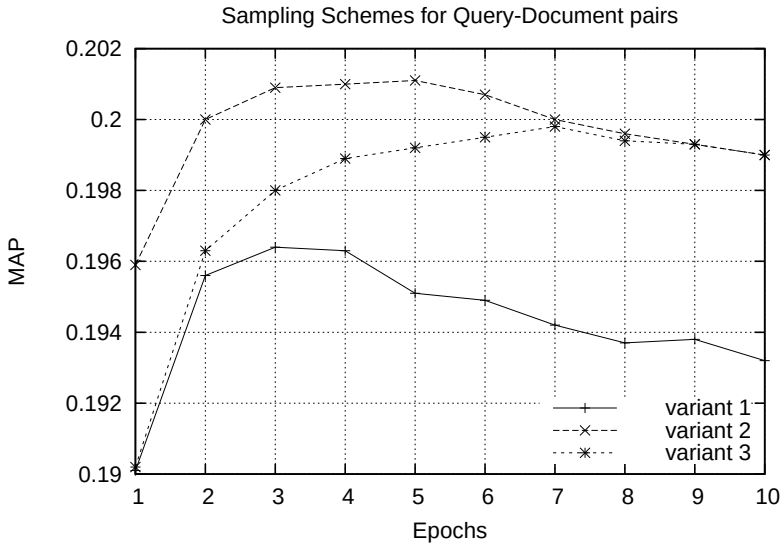

Figure 5.6: Influence of different query-document-document sampling schemes on MAP score with regularization parameter $\lambda = 10^{-8}$.

Feature Selection

The previous experiment on stop word filtering has already shown that feature selection based on ℓ_1-regularization helps to improve generalization capabilities. So, in contrast to the same experiments on the 20NG corpus (see Section 4.2), this time regularization actually helps. It was already suggested in Section 4.3 that retrieval on the 20NG corpus is an extraordinary task because of the specific properties of the newsgroup data.

This time again, we are using the DF-tables from the training set to construct the development and test set data and we observed an increase in score. The argument of conserving the same distribution does still apply, although the vector binarization step makes this argument less convincing.

Enforcing Sparsity

In general, the set of all possible word pairs contains a lot of noise. This is even more true here, because the word patterns inside the two languages are completely different. Applying an ℓ_1-regularizer that actively pushes low values to 0 enforces sparsity of the model and helps to give more weight to useful word correlations, increasing generalization capabilities of the final model.

Figures 5.2, 5.3 and 5.4 illustrate that if the parameter is chosen wisely, regularization improves the model significantly and the regularized model has a clear advantage over other models in terms of generalization capability. To see the amount of sparsity actually achieved, Table 5.3 lists the sizes of the models from the comparison. The last row contains the size

epoch	1	2	3	4	5	6	7	8	9
sampling 1	57.7M	28.1M	18.1M	13.0M	10.2M	8.3M	7.1M	6.1M	5.3M
sampling 2	47.0M	23.7M	15.7M	11.5M	9.0M	7.4M	6.2M	5.3M	4.7M
sampling 3	25.6M	10.2M	5.7M	3.7M	2.7M	2.0M	1.6m	1.3M	1.2M
$\lambda = 0$	2,402.4M								

Table 5.3: Model sizes indicating sparsity of differently trained models. Numbers are in megabytes (10^6 bytes), ℓ_1-regularization parameter is $\lambda = 10^{-8}$ for all three variants.

of the unregularized model as an "upper bound". A notably observation is the fact that variant 3, where each query is paired with all relevant documents and random irrelevant ones, has not only the highest score but also results in the smallest model.

It should be emphasized that we were only training on 5 million instances and it is possible that increasing the training data changes this picture in favor to lower λ-values. In the following experiments, we use a regularization parameter of $\lambda = 10^{-8}$ unless otherwise noted.

Reducing Feature Space

For feature space reduction we examined the same strategies as for monolingual data. Thus, we are conducting experiments that basically aim at analyzing the impact of the filter methods limited vocabulary and correlated feature hashing.

Limited Vocabulary

The average length of documents of the unmodified patent corpus is 178.8 words on average for Japanese compared to 134.8 for English, and the size of the vocabulary is 60,470 distinct words for the Japanese side and 365,553 for the English side (see Table 5.1).

This discrepancy in counts is a pointer to the fundamental differences between the two languages. One reason for the huge difference in raw counts is the fact that Japanese *kanji* characters have multiple meanings depending on the context. Those characters were imported from China, and in some cases multiple times under different circumstances. The existing meanings were preserved and new senses added, thus written Japanese today is a very condensed and efficiently encoded language.

Limiting the vocabulary is a way primarily to reduce model and training complexity. In the case of a hashed parameter vector, limiting the vocabulary also reduces unwanted collisions and thus decreases this kind of information loss. In our experiment we applied a limited dictionary of $30,000$ words in the way described in Section 3.2.

The expectation was that a simple action like cutting off the dictionary at 30,000 words must degrade performance significantly, but the experiment's result were different. As Table 5.4 demonstrates, the scores are almost exactly on par with our best model so far, i.e. the model using implicit stop word filtering and the new sampling scheme, which was also

epochs	1	2	3	4	5	6	7
standard	0.1959	0.2000	0.2009	0.2010	0.2011	0.2007	0.2000
model size	25.6M	10.2M	5.69M	3.71M	2.67M	2.04M	1.62M
limited	0.1952	0.2002	0.2009	0.2010	0.2008	0.2005	0.2002
model size	17.9M	7.17M	4.08M	2.69M	1.92M	1.49M	1.23M

Table 5.4: MAP scores and model sizes for two models using full and limited vocabulary. The ℓ_1-regularization parameter is set to $\lambda = 10^{-8}$ in both cases.

used for creating the data with the limited vocabulary. The model sizes, however, are much smaller because the limited dictionary leads to less collisions and noise. With less noise, the algorithm can better extract useful patterns, although the process of reducing the dictionary also means information loss.

One important aspect is that the amount of reduction is not even between the two languages. For the Japanese side, about 50% of the types are discarded, for the English side approximately 92% are removed. So, using the described limiting approach, there is much less information removed from the Japanese than from the English texts. After the reduction the algorithm might be able to connect Japanese to English words better because some useful patterns in the patents are still preserved.

Number Normalization and Simplification

The idea behind number normalization is to convert any value into a format that can be compared among each other. For example, 1.0 and 1 describe the same number, but a term-based information retrieval system will normally fail to recognize this identity.

Correct and reliable normalization of numerical values is a complex task. For example, imagine the memory capacity of a computer device is "256 GB". Depending on the type of device, this number has to be converted differently: in the case of memory modules its $256\ \text{GB} = 256 \times 1024^3 = 274,877,906,944 \approx 2,7488 \cdot 10^{11}$ bytes, in the case of hard drive space its approximately $2,56 \cdot 10^{11}$ bytes. There are too many complicated aspects and for this reason we decided not to put too much effort in normalizing numbers correctly.

We implemented a "poor-man's-normalization" scheme based on simple regular expressions. The method is to replace all digits by a special token. The capital "N" is used for this, because the rest of the text is lowercased. For example, "128" and "999" will both be replaced by "NNN", which might not seem useful at first. However, we believed that this conversion increases overlap between relevant patents, because such patents are likely to mention similar types of numbers. From this perspective, this simplification step is a reasonable strategy to improve coverage.

The result of the trade-off experiment comparing information loss and higher overlap by number simplification can be found in Figure 5.7. Number simplification actually helps and increases scores compared to the best model so far (standard model with $\lambda = 10^{-8}$). Although the increase is low, we found that the difference between the 5th epoch of the

standard model (MAP = 0.2011) and the 4th epoch after simplification (MAP = 0.2055) is significant according to the randomization test described in Section 3.3.

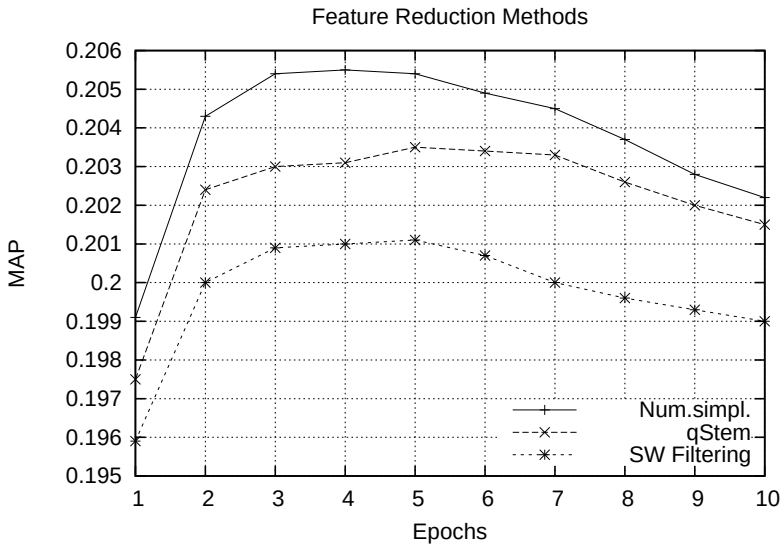

Figure 5.7: Comparison of different feature reduction methods. Number simplification and stemming both improve MAP, but for qStem the score change is only significant at $p = 0.05$.

Stemming and Lemmatization

The original idea was to leave as much as possible of the data untouched and let the machine learning select informative terms or pairs. Different inflections of verbs and declensions of nouns and adjectives will be identified and grouped by the learning algorithm as long as enough data is available. The previous experiment on number simplification already suggested that simplifying the data supports identification of useful terms.

Lemmatizing is a standard approach in information retrieval to remove noise and increase coverage between terms. The idea is to reduce every grammatical form to the most basic form, its *lemma*. However, lemmatizing is a difficult task: a lot of linguistic effort has to be put into a tool that reliably generates the lemma for a given word, because sometimes the word can be of different types and must be disambiguated by the context or the syntax structure.

A much easier approach is stemming, where the grammatical forms are simply reduced to the stem of the original word. For the English language, an efficient implementation can be achieved by applying a few rules based on regular expressions. A well known example that works on English texts is the Porter stemmer (Porter, 1980). Although the quality of the result is often bad from a linguistic perspective, we look at this idea as a feature reduction technique to increase overlap.

	punct. filt.	+ TFIDF filtering	+ Number simpl.	+ qStem
∅ vector length$_{jp}$	67.15	50.71	49.23	41.53
$\mathcal{D}_{jp}$	59,410	59,389	57,459	50,342

Table 5.5: Average length of documents and vocabulary sizes after several filtering steps.

Before explaining the stemming approach we applied, a brief explanation of the three character systems in written Japanese is helpful. Japanese written language consists of three types of characters: *hiragana* (ひらがな), *katakana* (カタカナ) and *kanji* (漢字). It was already mentioned that *kanji* are "imported" Chinese characters, so these are usually content words. Function words are always written in *hiragana* and also grammatical modifiers that are attached to content words. The remaining group, *katakana*, is reserved for words imported from language families other than Chinese. For example, "*kon-pyuu-ta*" (コンピュータ) is the English imported word for "*computer*".

Japanese is a agglutinative language, where verbs show a high degree of inflection. Adjectives are also highly inflected, but nouns only very little. For example, the plural of a noun is only identified by the context. Finding the correct lemma for a given Japanese word is a task we never considered seriously, but we thought of a very simple method based on the characteristics of the Japanese writing system. As practically all function words and grammatical forms are written in hiragana and content words are (almost) always in katakana or kanji, a radical approach is to remove hiragana characters completely from the text. In the following, we call this harsh form of stemming *qStem* ("quick'n'dirty stemming" or "query stemming").

Deleting all hiragana characters changes statistics on document length and vocabulary. These changes after subsequently applying several filtering steps are listed in Table 5.5. The method qStem has a high impact on document length and vocabulary size. All in all, the document length changes by almost 40%, while the vocabulary is reduced by 16%. Thus, while the learning complexity is significantly reduced, we hope to keep the most "informative" part of the vocabulary.

It must be noted that hiragana words are not always functional words and grammatical endings. In fact, any kanji character can also be written in hiragana and in rare cases, e.g. where the sense is clear and the kanji character is very complex, hiragana is preferred. These words, although they might be useful, get discarded during the filtering as described above.

From a linguistic point of view, this filtering is a questionable method, nevertheless the results in Figure 5.7 show that this form of lemmatizing helps to increase MAP for the retrieval task. However, the gain over the standard method is low and is in fact only significant at $p = 0.05$, thus not clearly supporting a definite positive contribution.

Correlated Feature Hashing

Correlated feature hashing has been shown to be very effective in the monolingual experiments in chapter 4. For the monolingual task, the biggest advantage is the decrease of

word	mappings				
supersampler	interpolates	interpolator	greatest	interpolating	offsets
N.N-N.Nk.sub.N	NN.N-NN	as.sub.N	wo.sub.N	ceo.sub.N	sno.sub.N
aminoglucoside	antibiotic	liposome	self-regulating	ineffective	toxic
suspend-to-disk	reload	reboot	routines	save	saved
cracks	cracks	crack	cracking	cracked	scratches

Table 5.6: Sample mappings generated by correlated feature hashing on patent abstracts.

features while preserving information contained in the features. Additionally, hashing into multiple bins minimizes the harm of collisions.

For the cross-language task, it was not clear if these two properties contribute in the same way. Here, the system not only learns the characteristics of the information need, but also an implicit translation table of word pairs from different languages. By increasing the number of words per document, there are more possibilities to connect source and target words, making it more difficult for the model to extract information.

In all previous experiments on the cross-language task, reducing complexity helped the model to learn useful patterns. Even in the case of limiting the vocabulary, the model was on par with the standard model, and if the information had been preserved, it would have worked even better. Thus, we decided to first apply the CFH-algorithm with a mapping of 30,000 words to the English side only. On the Japanese side with approximately 60,000 words, we applied all the described filtering steps except qStem.

Due to memory limitation on the Hadoop-cluster, it was necessary to use not more than 29 bits to address the parameter vector, which results in a 4 GB matrix of approximately $23,000 \times 23,000$ size. Exactly $536,870,912$ buckets are available for features, that are in the case of the full vocabulary $365,553 \times 60,470 = 22,104,989,910$ possible word pairs. This means $41,17$ collisions per bucket on average, which explains why the standard model's performance is so low.

With correlated feature hashing for English and filtering of the Japanese side the number of possible word pairs is reduced to $30,000 \times 57,459 = 1,723,770,000$ resulting in an overload factor of 3.21 for each bucket on average. This is still not optimal, but we hoped to see improvements under optimized regularization settings. Again, we generated the mappings on the training set and applied them on all sets, effectively filtering out all words that did not occur in the training set and hashing ("mapping") the remaining words to the 5 most correlated buckets ("frequent words").

Table 5.6 lists some example entries from the CFH tables created on the English documents of the patent data. Compared to the examples in Table 4.8 on the 20NG corpus, these entries make more sense. There are still some strangely mapped terms, but for most of the words the CFH works. The example in the second row contains the mapping of a complex molecule structure to simpler components, and demonstrates exemplary the results of number simplification. The last line is the mapping for the word "cracks" which is a

frequent word itself. This mapping is well understandable and makes perfect sense.

As for the 20NG corpus, correlated feature hashing changes the length of the documents significantly. After this mapping, the length of the documents grows by a factor of 5, but there are frequent repetitions and after stop word removal (Section 5.2) the length of the documents grows roughly by a factor of 3. This fundamental change in data characteristics requires a new tuning of the ℓ_1-regularization parameter λ. We searched for the best λ in log space and compared the results to the graph of the best model so far, which is the standard model with number simplification from Section 5.2.

The result of the search for λ is shown in Figure 5.8. The results are encouraging, because correlated feature hashing actually boosts MAP by more than 2 points. Please remember that the standard model gets slightly better after 4 runs over the data, but in the scale of the figure this increase is very small.

As we are getting our system in shape for the final evaluation, we also compared the PRES scores, our final metric introduced in Section 3.3, to select the optimal system to test on. This time we compared the two best systems (Figure 5.9). The behavior of the PRES scores is almost the same as MAP, but there are differences in detail. For MAP, the optimal regularization parameter lies between $7 \cdot 10^{-9}$ and $8 \cdot 10^{-9}$. For PRES, the optimal regularization parameter is $\lambda = 9 \cdot 10^{-9}$ and the shape of the curve is much steeper, making it easier to identify the optimum. Interestingly, at the lower left both models perform similar according to PRES, while the model without stemming performs better according to MAP. With increased λ this preference switches for MAP, but PRES seems more stable in promoting the variant with qStem.

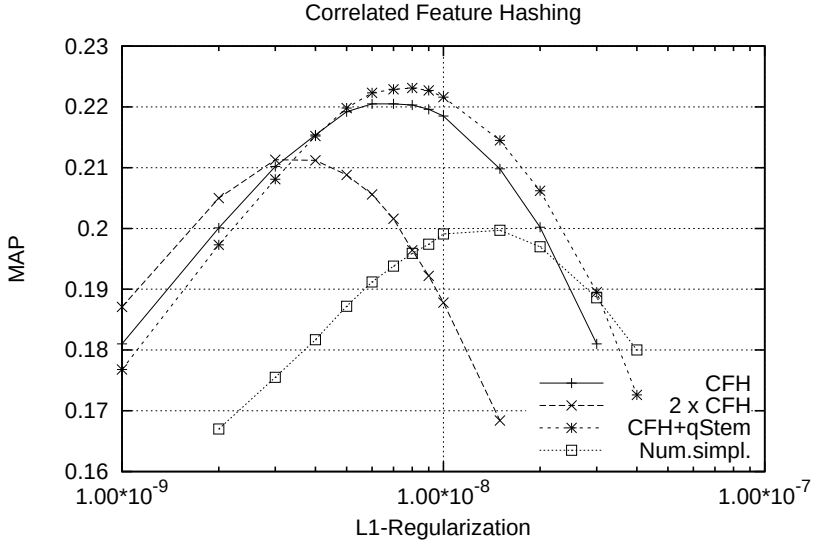

Figure 5.8: Searching for the optimal λ according to MAP.

Until now, we had applied CFH only to the English side. One question was whether CFH can further increase the retrieval performance when applied to both sides. For this to find

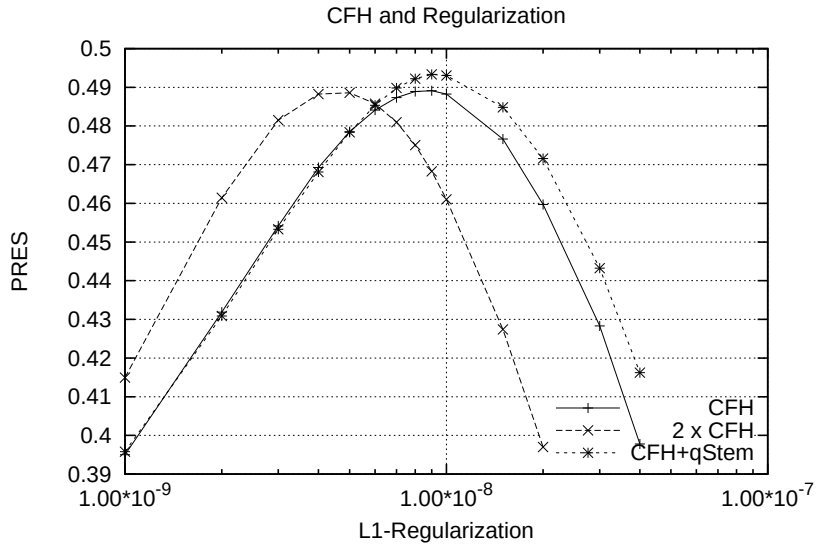

Figure 5.9: Searching for the optimal λ according to PRES.

out, we created mappings for the Japanese side in exactly the same way as for the English side, i.e. created the mappings on the training corpus with $k = 5$ and $\mathcal{F} = 30,000$, and applied these mappings to training, development and test set.

This mapping of course increased the length of Japanese documents by a factor of approximately 3. This time, the results in Figure 5.9 showed no increase for PRES and in Figure 5.8 a huge decrease in MAP. Our explanation is that CFH on one hand reduces the vocabulary, but on the other hand increases the number of features per document by increasing the document's length. Specific words might be good indicators for relevance, pushing certain documents very high. With CFH, the impact of such terms is softened, resulting in overall lower precision but higher recall.

The final step would be to find out the optimal number of training instances that gives the best performance before overfitting. We learned multiple passes on the data with settings for the best model so far, i.e. CFH for English and qStem for Japanese. The result was that performance degrades slightly after the second epoch, indicating that overfitting already started. We then tried to learn on data splits, but the result was even worse, most probably due to a technical problem with the learning toolkit.

Our explanation for the early overfitting of the best model is that by increasing the number of words per document by a factor of 3 (as CFH does), the system actually sees three times more data as in the non-CFH case. As a simple rule of thumb, overfitting should then occur three times earlier, which means according to Figure 5.7 approximately after $\frac{4}{3}$ of the original data. So, we conclude that after the first epoch we are already relatively close to the optimum and stopped further investigations in this direction, accepting that we loose a few additional points here.

Intermediate Results

We will now briefly summarize the evolution of our sparse model documented in Table 5.7. These scores are again discussed in the final section of this chapter.

In the first experiments we discovered the problem of using TFIDF weights in our task. By moving to binarized vectors, the model started to extract useful patterns. For further improvements, ℓ_1-regularization had to be applied to actively remove irrelevant features. Still, the model trained on the completely unmodified data had problems.

Our TFIDF-motivated method of stop words removal increased PRES and MAP scores over the unmodified model significantly. A somewhat unexpected result was the positive effect of the sampling scheme on the learning performance. We moved from completely random query-document-document training instances to sequences of tuples that contained all relevant documents for a certain query.

Simple variants of classic IR filtering methods such as stemming and number simplification further improved scores, but the change was not always significant. Besides stop word filtering, CFH made the largest contribution to the final scores. For the CFH model with number simplification, additionally applying the qStem method finally showed a significant change.

	PRES		**MAP**	
	dev	**test**	**dev**	**test**
unmodified	0.3315		0.1765	
TFIDF stop words	†0.4196		†0.1927	
new sampling	†0.4252		†0.2011	
qStem	†0.4452		0.2035	
Num.simpl.	‡0.4481		‡0.2055	
CFH+Num.simpl.	†0.4882		†0.2185	
CFH+Num.simpl.+qStem	†0.4930		†0.2216	
final model $\lambda = 9 \cdot 10^{-9}$	**0.4933**	0.4994	**0.2227**	0.2301

Table 5.7: Intermediate and final results documenting the evolution of the sparse model. Changes preceded by (†) are significant to the previous row, (‡) indicates a significant change over the second previous row.

Concluding this section, the best model so far uses CFH on the English side, qStem on the Japanese side, and is trained on approximately 5 million instances with an ℓ_1-regularization parameter $\lambda = 9 \cdot 10^{-9}$. This is the model we use for further experiments in the rest of this chapter.

5.3 Combining Sparse and Dense Models

The final approach presented in this thesis is a novel method where we combine the result of the sparse model with dense features. The basic idea is to take the scores of the best model of the previous Section 5.2, add dense features that we think of to be useful, and train a dense model on the newly created data. Combining different rankings has been done various times with different strategies in mind. It is the central idea of *rank aggregation* which was shortly mentioned in the introduction in chapter 1. The new approach combines dense and sparse features by "plugging" the output of the sparse model as a dense feature into the new model. In this new model, we do not simply re-rank the top-k ranked outputs of several ranking systems as in standard rank aggregation applications, but we have access to the full list of documents and use this information for learning.

Dense Features

We take the score of our sparse model as a feature and add domain knowledge features inspired by the work of Guo and Gomes (2009). The first step is to define a feature map that takes pairs of patents (p_i, p_j) and returns the dense features we want. Formally, the resulting vector Φ can be constructed as follows:

$$\Phi(p_i, p_j) = (\phi_1(p_i, p_j), \ldots, \phi_n(p_i, p_j))$$

Each ϕ_k, with $k = 1, \ldots, n$, is a function that returns a single dense feature, for example the score our sparse model assigns to the pair (p_i, p_j). This way we can easily add or remove features that we think of to be useful. After training, we can compare the learned weights to find out which features are most or least useful for the task.

Score Feature

The score feature does not need further explanation. We simply take the best sparse model so far (see Section 5.2) and define our function according to the basic model:

$$\phi_{score}(p_i, p_j) = \mathbf{p}_i^\intercal \mathbf{W} \mathbf{p}_j$$

By definition, the function ϕ_{score} returns a real valued score for each patent pair (p_i, p_j).

Binary Features

The remaining features are binary values defined on common properties of the two patents. A short description of each feature follows here:

- *Same IPC class*: every patent has assigned one or more IPC codes (see Figure 5.10 for an example). For this feature, we compare the first four characters of the code, i.e.

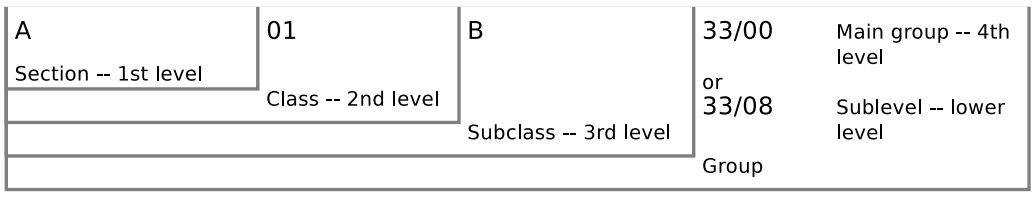

Figure 5.10: Example of the complete classification symbols `A01B 33/00` and `A01B 33/08` combining codes for section, class, subclass and main group or subgroup. Idea from: "International Patent Classification (Version 2013)", WIPO.

the section, class and subclass code, and call this function IPC_3. If all three codes are equal between p_i and p_j, this feature becomes active.

$$\phi_{ipc3}(p_i, p_j) = \begin{cases} 1 & \text{if } \text{IPC}_3(p_i) \cap \text{IPC}_3(p_j) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

This is certainly a strong feature: being in the same IPC-section, class and subclass should raise the possibility of relevance.

- *Common inventor*: patents have at least one inventor defined. This feature looks at the intersection of the inventors of patents p_i and p_j. If there is at least one common inventor, this feature is 1, otherwise 0.

$$\phi_{inventor}(p_i, p_j) = \begin{cases} 1 & \text{if } Inventors(p_i) \cap Inventors(p_j) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

The motivation for this feature is that an inventor is likely to make a new invention based on his previous work.

- *Assignee*: this is the legal owner of a patent, called *assignee* in the U.S. and *applicant* elsewhere. This feature compares the contents of the field named "applicant" in the patent data.

$$\phi_{assignee}(p_i, p_j) = \begin{cases} 1 & \text{if } Assingee(p_i) = Assingee(p_j) \\ 0 & \text{otherwise} \end{cases}$$

The assumption is that patents belonging to the same assignee are more likely to be relevant to each other.

- *Independent applicants*: this feature captures if both applicants of the two patents are independent.

$$\phi_{independent}(p_i, p_j) = \begin{cases} 1 & \text{if } IndependentApp(p_i) = IndependentApp(p_j) \\ 0 & \text{otherwise} \end{cases}$$

One possible scenario for this feature is that patents by independent applicants are more likely to build up on other patents by independent applicants.

- *Claims*: This feature compares the number of claims the patents make and fires if the number lies in some common interval.

$$\phi^n_{claims}(p_i, p_j) = \begin{cases} 1 & \text{if } |Claims(p_i)| \in C_n \wedge |Claims(p_i)| \in C_n \\ 0 & \text{otherwise} \end{cases}$$

  The intervals are $C_1 = \{0\}$, $C_2 = \{1, \ldots, 5\}$, $C_3 = \{6, \ldots, 10\}$, and $C_3 = \{11, \ldots, \infty\}$, defining four binary features in total.

  Claims define the boundaries of patent protection. They follow a certain structure describing the underlying invention in a precise language often using legal terms. Thus, a possible motivation for this feature is: two patents that both contain many claims are more likely to be relevant to each other than two patents that made only few claims.

- *Geographic location*: The motivation of this feature is to infer information about relevance based on geographic data on the first inventors of both patents.

$$\phi_{[f]}(p_i, p_j) = \begin{cases} 1 & \text{if } Address[f](p_i) \cap Address[f](p_j) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

  In this notation, $Address[f](p)$ returns the address field f of the inventor of patent p, where $f = \{\text{Country}, \text{State}, \text{City}\}$. This gives a total of three binary features on geographic information.

  The idea behind this feature is that two inventors working at the same place are more likely to cite each other's patents than inventors working at different locations.

  The set-style definition of common address fields is due to the fact that we had to keep multiple addresses for a single inventor, so $InventorAddr(\cdot)$ actually returns a set. More information on this can found in the next section on the construction of features.

- *Sections and Classes*: The idea for these newly added features comes from the assumption that the IPC_3 feature might be too strict in comparing always the first three elements of the IPC-code. By looking also at IPC sections and classes, this feature enables a finer evaluation of the code.

$$\phi^n_{ipc[1,2]}(p_i, p_j) = \begin{cases} 1 & \text{if } |\,\text{IPC}_{[1,2]}(p_i) \cap \text{IPC}_{[1,2]}(p_i)| \in P_n \\ 0 & \text{otherwise} \end{cases}$$

  The used sets are $P_1 = \{0\}$, $P_2 = \{1\}$, $P_3 = \{2\}$, $P_4 = \{3, \ldots, \infty\}$. The notation is similar to the first IPC-feature: $\text{IPC}_1(p)$ is the set of *sections* of IPC-codes for patent p, $\text{IPC}_2(p)$ the set of *sections and classes*. This returns four features for each of the two components, i.e. 8 in total.

To summarize, this gives 19 binary features plus one score feature resulting in a total of 20 dense features.

Construction

Most of the data required to construct the dense feature map ϕ is available in the MAREC patent corpus. Having this data, the function ϕ_n simplifies to a comparison or intersection creation of corresponding entries.

However, the locality features are not easily accessible and are completely missing in the Japanese patents we use. Our solution was to create an inventor's address database based on the data from MAREC. This data structure, actually a hash table, maps inventor names to corresponding addresses. For this task it was very helpful that all data fields contain only latin characters, so we hoped that our comparisons are relatively reliable. However, during the implementation of the database, we observed numerous inconsistencies in the address fields. For U.S. patents, the addresses are mostly error free, but the list of problems with foreign patent addresses is endless.

An exemplary problem is the *city* field for Japanese addresses. Here are a few examples of the variants found for the city name *Kyoto*:

- Most often, "Kyoto" is entered. This is what we hoped to always encounter.
- "Kyoto-shi" is also frequently used. This is actually the full Japanese name for "City of Kyoto".
- Sometimes, "Kyoto-fu" is found, which is the prefecture of Kyoto. This is a large area of $4,612$ km^2 including the city of Kyoto.
- Other patent applicants translated the Japanese name, providing "Kyoto City" or "Kyoto-city" for the city field.
- A difficult example is the specification "Fushimi-ku", which is actually a ward of Kyoto, so it is part of the city of Kyoto.

These are some of the problems encountered on the spelling of a relatively famous Japanese city. We found more irritating examples where *city* fields contain almost complete addresses, e.g. "Kaminoge, Setagaya-ku, Tokyo": it is the area *Kaminoge* in the *Setagaya* ward of *Tokyo*. A deceitful example is "Minato-ku", which can be one of the 23 wards of Tokyo, or it can be a ward of Osaka.

We expected less problems with German addresses, but our hopes were dashed: we counted more than 40 different variants to spell the city name "Munich". Several problems are due to the missing characters on the English keyboard, so "München" is usually entered as "Munchen", but surprisingly, "Munich" sometimes becomes "Münich" in the data.

To come to an end, we implemented a set of approximately 100 rules of regular expressions that corrected the most common errors found in city names. If multiple specifications were encountered like in the Japanese example above, they were broken up into multiple addresses. In the example above, we would create three city specifications, i.e. "Kaminoge", "Setagaya-ku", and "Tokyo" – geographically wrong, but we aimed for coverage.

To lower the impact of the problems mentioned above, we finally implemented an inexact comparison algorithm based on Levenshtein-distance (Levenshtein, 1966). The Levenshtein-algorithm returns the *edit distance*, i.e. number of basic operations to convert a source string s into a target string t. The basic operations are substitution and insertion. For our comparison, we defined that two city names match if $0 \leq \mathit{edit\_distance}(s,t) \leq 1$, but this strategy punishes longer names. As additional match criterion we defined $\mathit{edit\_distance}(s,t)/\min(|s|,|t|) \leq 0.2$. This means, for longer strings we allow an edit distance of 0.2 times the length of the shorter of both strings. For example, a string with 10 characters can match a 12 character string if the edit distance is not greater than 2.

Learning with Dense Features

Learning with dense features is very similar to learning with sparse features. We took the same learning toolkit as before, Vowpal Wabbit, and fed it with differences of dense feature vectors together with relevance judgments to train a ranking perceptron.

We first created pairs of higher and lower ranked English patents with respect to a given Japanese patent, $x = (\Phi(p_{jp}, p^+_{en}), \Phi(p_{jp}, p^-_{en}))$, where p^+_{en} is ranked higher than p^-_{en}. For convenience, we define the difference between the two feature vectors $\overline{\mathbf{x}} = \Phi(p_{jp}, p^+_{en}) - \Phi(p_{jp}, p^-_{en})$.

This time there are 20 dense features without any quadratic expansion, so the objective function simplifies to a perceptron for pairwise ranking (Shen and Joshi, 2005). In our case, we use a margin ranking perceptron as presented by Collobert and Bengio (2004):

$$\min_{\mathbf{w}} L(\mathbf{w})$$

$$\begin{aligned} \text{where } L(\mathbf{w}) &= \frac{1}{N}\sum_{i=1}^{N} l_i \\ &= \frac{1}{N}\sum_{i=1}^{N} (b_i - \mathbf{w}^\intercal \mathbf{x}_i)_+ \end{aligned} \tag{5.2}$$

This is exactly the optimization problem as we have described in Section 2.2. Again, we use stochastic gradient descent as learning algorithm. According to Equation 5.2, the loss function on a single example is

$$l_i(\mathbf{w}) = (b_i - \mathbf{w}^\intercal \mathbf{x}_i)_+$$

from which the following SGD update rule can be derived exactly as for Equation 2.9:

$$\nabla l_i(\mathbf{w}) = \begin{cases} -\overline{\mathbf{x}}_i & \text{if } \mathbf{w}^\intercal \mathbf{x}_i < b_i \\ 0 & \text{otherwise} \end{cases}$$

Everything left to do is to create learning data for the tool and start training. For the score feature, we use Vowpal Wabbit's code as a library, which is actually one of the key features of VW. Technically, we create a program instance in memory with the appropriate parameters and the model, and can then send vectors and receive scores comparable to a client-server model. Given a tuple $(p_{jp}, p^+_{en}, p^-_{en})$, where p^+_{en} is a more relevant and p^-_{en} a lesser relevant patent with respect to p_{jp}, we create two scores $\phi_{score}(p_{jp}, p^+_{en})$ and $\phi_{score}(p_{jp}, p^-_{en})$ which fill the corresponding dimension of the two feature vectors $\Phi(p_{jp}, p^+_{en})$ and $\Phi(p_{jp}, p^-_{en})$.

To add the domain knowledge features to Φ, we implemented an in-memory database which is based on hash tables for fast access. The database expects a patent ID and the field-type as parameter, and returns the requested meta-data. Then, our code only needs to compare fields or build intersections of sets and count the elements.

The geographic location are the remaining features of Φ and these were difficult to implement. As mentioned before, we had to cope with various issues related to the address data. The application of the Levenshtein-algorithm for the comparison of city names catches some errors, but we were not sure if the additional information will make up for the noise we introduce with this feature.

Experiment

In the final experiment we evaluate the model using all features described above and also do an ablation experiment where single features (or feature-groups) are left out for both training and testing to see its contribution.

The results of the dense model in comparison to the sparse model are listed in 5.8. Unfortunately, we cannot directly compare these results to the work by Sokolov et al. (2013), because we use different test-dev-splits: we started tuning our experiments on the test partition and thus later had to switch development and test set, so we were not forced to repeat all experiments that were done before.

	PRES		**MAP**	
	dev	**test**	**dev**	**test**
sparse model	0.4933	0.4994	0.2227	0.2301
dense model	**0.6933**	**0.6881**	**0.2891**	**0.2901**

Table 5.8: MAP scores of the two final models based on sparse and dense features.

However, the numbers for the new method are encouraging and in Section 5.3 we re-tune our model and provide an argument under which circumstances the numbers are comparable to the results of Sokolov et al. (2013) to some extent.

Contribution of Single Features

To determine the contribution of each dense features, we conducted an ablation experiment where our dense model is trained and tested without the features we want to examine. The

idea is that an ablation experiment gives us important information how much a feature contributes to the patent CLIR task.

feature	**weight**	**PRES**	**MAP**
all	–	0.6933	0.2891
ϕ_{score}	1.411876	†0.4861	†0.1871
ϕ_{ipc3}	0.945277	†0.6507	†0.2816
$\phi_{inventor}$	0.239652	†0.6907	†0.2773
$\phi_{assignee}$	0.017165	†0.6929	0.2888
$\phi_{independent}$	0.043821	0.6934	0.2891
ϕ^1_{claims} (0)	0.051620		
ϕ^2_{claims} (1-5)	0.024788		
ϕ^3_{claims} (6-10)	-0.000341		
ϕ^4_{claims} (11+)	0.021732	0.6932	0.2889
ϕ^1_{ipc1} (0)	0.000417		
ϕ^2_{ipc1} (1)	0.079988		
ϕ^3_{ipc1} (2)	-0.025161		
ϕ^4_{ipc1} (3+)	-0.100250	0.6933	0.2890
ϕ^1_{ipc2} (0)	-0.278392		
ϕ^2_{ipc2} (1)	0.003336		
ϕ^3_{ipc2} (2)	0.210527		
ϕ^4_{ipc2} (3+)	0.406972	†0.6881	†0.2843
ϕ_{city}	0.057727		
ϕ_{state}	-0.016517		
$\phi_{country}$	-0.022161	†0.6935	0.2897

Table 5.9: Weights of features and the influence of features / feature groups on the scores for the development set. Values preceded by (†) indicate a significant change compared to "all" system.

The creation of the training data for this task is straightforward: we have already the dense data, so we just need to switch off certain features in the training set. This can be accomplished by restricting training to certain namespaces of the data, or simply by creating new sets where the features in question are left out. The latter approach, which we followed here, has the nice property of generating models that are compatible to our testing setup.

This property comes from the fact that the new models simply have zero weight at the position of the left out feature, thus we can use *exactly* the same test setup as for the normal dense model. This makes the execution of the single experiments very comfortable.

Table 5.9 lists the results of the experiment where we switched off each single feature described in Section 5.3. Additionally, the table contains the weights for each feature after training the normal dense model (row "all"). According to the test described in Section 3.3,

most features are significant, at least according to one of the two metrics. Only the changes for the features $\phi_{independent}$, ϕ_{claims} and ϕ_{ipc1} do not indicate a significantly different system.

Looking at the weights, the ϕ_{score} feature is by far the most important, especially because this real value is not normalized and lies roughly between 4 and -2. The PRES and MAP values in row ϕ_{score} are the model's scores *without* the sparse system. Compared to our sparse model, the impact of the domain knowledge features is impressive. A model trained on these features alone almost reaches the PRES scores of the final sparse model and is better than the unmodified sparse model with respect to MAP.

The second most important is the ϕ_{ipc3} feature. It is quite reasonable that having a common IPC-Code in section, class and subclass is a high indicator for relevance, because then both patents are basically covering the same area. The next level would be the group category, which is probably too fine grained – but this is speculation, we did not create a feature to verify this.

The third most important is the ϕ^4_{ipc2} feature, where a correlation in section and class for more than 3 IPC-codes is a strong indicator of relevance. However, this might be a rare case as we have not checked how often this happens in the data. The feature for having no common IPC-codes that match in section and class has the highest negative weight. This is a strong indicator that two patents are coming from completely different areas. Nevertheless, the weights for the features ϕ_{ipc1} that fire when 2 or more common sections occur is negative too, which suggests that common section membership is in general not a good indicator for relevance, because the classes contained in a section are often too diverse[5].

Having a common inventor is by its weight a strong feature too. This supports our initial assumption that an inventor is indeed likely to build on his previous work and to cite his own patents.

The contribution of the geographic location features, ϕ_{city}, ϕ_{state} and $\phi_{country}$, was disappointing, especially with the complexity of the construction in mind. However, the data we had to use to construct this features contains a lot of noise. Thus, it is likely if this feature is based on cleaner address data, it will contribute more to the task.

We only checked certain groups of features in this experiment. It would be interesting to conduct more experiments on the single features within a group to estimate their contribution to the task, but due to time constraints we had to move this to future work.

Incorporating New Models

In the final section about combining dense and sparse models, we take a different look on the model presented in the previous section. This model which we call the *dense model* for short, combines a sparse model together with dense features and learns a new model on the data. This combination is accomplished by putting the sparse model score as a feature into the dense model.

[5]For a complete list of current IPC-codes showing the diversity of classes and subclasses within a section, see `http://www.wipo.int/classifications/ipc/en/ITsupport/Version20130101/index.html`, accessed 02.09.2013

If we look at the dense features as single learners, the resulting model can be seen as an implementation of a multi-layer network, where sparse representations of data (words pairs, patent meta-data[6]) are the input data, the input layer emits dense features (IPC-information, addresses, scores) to the output layer, which finally generates the output. Figure 5.11 illustrates this perspective, where the sparse model is one component K_n of the input layer, the inputs x_n are documents, addresses, or meta data, and the single unit output layer M is our dense model.

The idea to treat features as (base) learners is a basic strategy of the boosting approach to machine learning (Friedman, 1999). In boosting, the method is to incrementally create a combination of base learners where the weight of each learner is adjusted by the residual of the current function. This clarifies why our approach cannot be considered a boosting method: we are not looking at residuals and we are not incrementally combining base learners. Our work is related to the method of linearly combining rankers as described in Wu et al. (2008). In this work, the authors directly optimize a non-smooth retrieval score like NDCG by training a LambdaRank model (Burges et al., 2006) using previous model scores as inputs. In contrast to this, we are optimizing the loss based on the relevance given by the data.

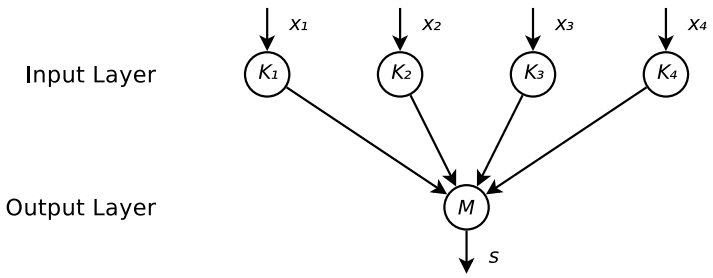

Figure 5.11: A different view of the combined model: multi-layer learning.

The network idea, however, inspired us to think of an extension of the dense model: the modularity of the approach enables us to plug in and out features (or learners) in a very flexible way. Thus, it is easily possible to add completely new modules of almost arbitrary complexity. One practical limitation of our current setup is that the new module should be written in a compiler-language to include the object code directly into the existing framework.

As a proof-of-concept of the idea presented above, we take the bigram model of the *linboost* implementation written by Sokolov et al. (2013) as a new module and use it as a feature in our dense model. The linboost approach applies the idea of boosting to CLIR and has performed exceptionally well on the patent retrieval task. The scores achieved by the single and the combined systems are compared in Table 5.10.

[6]If meta-data is represented by a binary vector, it *is* sparse.

Remarkable are the relatively high scores achieved on the precision based MAP metric. In this respect, our sparse model can beat the unigram-based Boost-1g model. However, for the recall oriented patent retrieval task, PRES scores are more important. In PRES metric, only our dense model can clearly outperform the unigram-based Boost-1g model and lies only slightly below the excellent performing bigram-based Boost-2g model.

The most interesting setup is of course the combination of our best model and the single best model by SOKOLOV et al. (2013). This combination shows the strongest performance for the PRES and MAP metrics and the changes over the plain Boost-2g and our standard dense model are both significant. This demonstrates that the idea of incorporating new modules actually works and that the system is able to combine "the best of both worlds".

	PRES		**MAP**	
	dev	**test**	**dev**	**test**
sparse model	0.4994	0.4933	0.2301	0.2227
Boost-1g	0.5850	0.6122	0.2064	0.1982
Boost-2g	0.6900	0.7196	0.2526	0.2474
std. dense model	0.6881	0.6933	0.2901	0.2891
dense w/ Boost-2g	†‡**0.7164**	†‡**0.7213**	†‡**0.3030**	†‡**0.3022**

Table 5.10: The final scores on the sparse model, the dense model and the combined model. Scores on Boost-2g were copied from SOKOLOV et al. (2013) for comparison. Values preceded by symbols indicate a significant change compared to the standard dense model (†) and to the Boost-2g model (‡).

Please note that in order to combine both models, we needed to optimize the sparse model on the same test set as the linboost model. As we had exchanged development and test set at the beginning, we needed to select the best model on *our test* for the combined model and evaluate the final on *our development* set. Fortunately, in both cases the model with ℓ_1-regularization $\lambda = 9 \cdot 10^{-9}$ performs best. As we were not able to optimize the number of training instances, the best sparse model according to the score on our test set is the same model as the one according to our development set. Thus, after exchanging test and development set we can use the same setup and same models to train and evaluate the combined system.

SOKOLOV et al. (2013) present a combination of scores based on a Borda Count method (ASLAM and MONTAGUE, 2001), which is according to the taxonomy in Figure 1.1 an unsupervised method for rank aggregation. Compared to their method, we are not restricted to top-k results. As long as k is relatively large and the number of combined modules small, this is practically not a limitation. At one point, however, the intersection of top-k results will become too small to learn weights that generalize well. In these cases, our model can perform better because it always takes the full list of retrieved documents into account.

5.4 Final Results

In this chapter, we started with the monolingual retrieval model from the previous chapter and transferred it into a cross-language setup. The advantage of the approach over other cross-language retrieval systems is the ability to learn word dependencies across languages in a pure machine learning setup. As the basic system is able to capture some important aspects of human language such as polysemy and synonymy, it represents an extremely flexible approach that can be applied to arbitrary languages.

In the first experiments we discovered that using TFIDF weighted vectors is suboptimal for the CLIR task we examined. After moving to binarized vectors, the model started to improve. Experiments on mixed vectors, i.e. binarized on one side and TFIDF on the other side, revealed interesting results. However, we did not find a direction for further experiments that indicated an advantage over the plain binarized system.

Unlike on the monolingual system examined in the previous chapter, ℓ_1-regularization has a positive influence on the cross-language model. Enforcing sparsity by ℓ_1-regularization not only keeps the model small but also supports the learning process by selecting useful features.

The TFIDF-motivated stop word removal increased PRES and MAP scores over the unmodified model significantly. Our explanation is that in our unigram-model setup, words with very low TFIDF-value must be considered noise and at the same time they become promoted by the binarization of the vectors.

An interesting result was the increased learning performance by exchanging the sampling scheme of query-document-document training instances. To our understanding, using a richer sampling scheme where sequences of tuples are created in a clever way that better supports the parallel learning of translation table and IR-similarity measure, which are both encoded simultaneously in the weight matrix M of the basic model.

Our method of number simplification had a surprisingly positive effect on the retrieval performance. This is due to the reduction in vocabulary, especially rare words, which not only decreases collisions in the hash table, but also increases coverage between near-similar terms such as parts of chemical formulas.

On top of these filtering methods, CFH with multiple binning to 5 bins made the largest contribution to the final scores. There are various reasons for this. CFH reduces the vocabulary, which actively reduces collisions in the parameter vector. At the same time, it increases the overlap between a query and a document by hashing to multiple bins – effectively, this a term-expansion on the query- and the document-side. Finally, CFH employs the hashing trick to reduce information loss originating from collisions in the hash table.

The last modification of the sparse CFH model with number simplification was adding the qStem method. In our previous experiments it was undecidable if qStem contributes to the scores. For the CFH model, applying qStem finally showed a significant change. After tuning the regularization parameter, we determined the final sparse model.

The final retrieval performance of our sparse system is on par with other unigram-based state-of-the-art systems. However, the unigram approach is limited. We think that for

	PRES		MAP	
	dev	**test**	**dev**	**test**
unmodified sparse model	0.3315		0.1765	
TFIDF stop words	†0.4196		†0.1927	
new sampling	†0.4252		†0.2011	
+ Number simplification	†0.4481		†0.2055	
+ CFH ($k = 5$)	†0.4882		†0.2185	
+ qStem	†0.4930		†0.2216	
best sparse model	0.4933	0.4994	0.2227	0.2301
domain knowledge feat.	†0.4861		†0.1871	
final dense model	**0.6933**	**0.6881**	**0.2891**	**0.2901**

Table 5.11: Intermediate and final results for the sparse and the dense model. Values preceded by (†) indicate a significant change to the previous row's system.

further improvements, the integration of higher order n-grams or the construction of other features that better capture language structure is necessary.

The dense model combines the score of the sparse model as a dense feature with domain knowledge features. These hand-crafted features defined on the patent domain showed a very strong performance for both PRES and MAP scores. A model trained on the 19 binary domain knowledge features alone almost reaches the PRES scores of the final sparse model and additionally beats MAP compared to the unmodified sparse model. However, the combination of score and domain knowledge features resulted in very high values for the PRES and MAP metric. Our interpretation is that both systems follow completely different approaches and their results are "linearly independent"[7]. Thus, the combination trained with machine learning techniques must give an increase in final performance.

Finally, we showed the versatility of our approach by integrating a completely different system as a new module. Retrieval results of the combined system showed that our dense system is in fact easily extensible and can make use of any system that generates ranking scores based on query-document pairs. The core component of the final dense model uses practically no memory, so limits on the number of modules are defined by the implementations of the single modules.

[7]This can be taken literally: if each query-document pair is a dimension and the rank is the value, then two systems generating different rankings are linearly independent.

Chapter 6

Discussion and Outlook

In this chapter we summarize the results of the experiments on monolingual and cross-language information retrieval. Section 6.1 will list the most important findings of our work and outlines strengths and weaknesses of our approach in a *discussion*. Section 6.2 is dedicated to *future work* that came across our mind during working on this thesis. In this last section, we talk about three aspects: First, *matrix decomposition* and *latent Dirichlet allocation* aim at feature reduction. Second, *distributed algorithms* and the implementation of *fast-response systems with MPI* describe ways to improve the system from a technical point of view. Third, *integrating other systems* continues the idea of combining different models in the dense model and mentions two concrete examples.

6.1 Discussion

In this work we examined the applicability of the basic model presented in Section 2.3 to a monolingual and to a cross-language retrieval task. In the monolingual setup evaluated on the 20NG corpus, the basic model had been applied in two different ways: learning the diagonal, i.e. a re-weighting of TFIDF values, and learning the full matrix. Compared to a plain TFIDF baseline, both models showed a strong increase in retrieval performance regarding the MAP metric, with the quadratic model performing much better than the diagonal model.

The quadratic model is able to learn word relations beyond direct matches, thus it is theoretically applicable in cross-language information retrieval task. Our experiments on the BoostCLIR corpus (see Section 5.1) showed that the basic model can be successfully applied to the CLIR task of patent prior art search across languages.

The quadratic dependency on vocabulary size of the basic system implicates among other things the problem of parameter complexity. We applied an approach presented by Weinberger et al. (2009) that hashes arbitrary large parameter spaces into smaller tables. Although this entails new problems such as information loss in case of collisions, it enabled us to use the full vocabulary for monolingual and cross-language experiments instead of a

limited dictionary.

Limited dictionaries are often used for retrieval experiments comparable to ours (Bai et al. 2010, Chen et al. 2010). However, our experiments in Section 4.2 showed that it is preferable to use the full dictionary in combination with hashing. In the monolingual setup, the positive impact of using all available data clearly surpasses the information loss that results from collisions in the parameter table. In the cross-language setup, the situation was undecidable: the experiment in Section 5.2 did not reveal a clear advantage for any of the two models. From a machine learning point of view, we opted for the full vocabulary and let the algorithm decide which features to use.

However, our experiments on the TFIDF weighting schemes revealed that collisions occur relatively often with a normal sized vocabulary. Section 4.2 showed that the increase in retrieval performance for the quadratic model by filtering out all words not present in the training data is substantial. The reason is that during training there is no statistical information available for these words, and during testing they might collide with existing entries. In a non-hashed setup, these words or word pairs simply get a 0-weight and do not contribute, but with parameter hashing they can collide with existing entries and thus contribute erroneously. Thus, in settings that use hashed weight vectors such a filtering step generally improves performance, especially if the hashed weight matrix is dense.

For the cross-language task, the TFIDF weighting was inapplicable. Experiments that examined one-sided non-binary weighting improved the results, but did note achieve the performance of binarized vectors. The implicit stop word filtering of TFIDF, however, has proven to be very useful for our task. This "TFIDF induced stop word filtering" significantly reduces noise in the data and helps to promote useful features, as we have examined in Section 5.2. In contrast to static stop word filtering through lists, it is able to adapt to certain properties of the data, e.g. particular words that are generally unusual but occur frequently in the corpus.

The biggest contribution to the retrieval performance of both systems came from the application of correlated feature hashing (CFH). We applied the variant with 5 bins in both cases, meaning that a single feature is hashed[1] to the 5 most correlated words in the corpus. Again, we observed a clear advantage when we create the mapping for both train and test on the training data only. An experiment with mappings created on the whole set was not conducted because in our opinion, this would carry over statistical information from the test to the training set.

There are various reasons for the positive influence of CFH in our setup. First, the method reduces the vocabulary and consequently reduces collisions in the hashed parameter vector. Second, it increases the overlap between a query and a document by hashing words to multiple bins; it was already mentioned in chapter 3.2 that this can be seen as a form of term expansion for information retrieval on either one or both sides. Third, CFH employs the hashing trick to reduce information loss because of parameter collisions in the hash table

[1] Although CFH creates a mapping, the subsequent hashing into the parameter vector makes the mapping behave like a hashing.

(Shi et al., 2009).

The method of ℓ_1-regularization for feature selection did not work equally well in both tasks. For monolingual IR, ℓ_1-regularization was unable to select useful features. With active regularization, the performance always decreases. This was unexpected and we believe that the reason lies in the data and the task. The 20NG corpus is probably too small and too specialized to benefit from this type of regularization. Nevertheless, it contains a lot of noise, e.g. spelling errors or colloquial language, and we expected ℓ_1-regularization to help reducing this noise, but this was not the case.

Other works such as the one by Chen et al. (2010) successfully applied ℓ_1-regularization to enforce a sparse model on the same dataset. In contrast to their results, our scores *without* regularization are significantly higher for the models with the learned diagonal and the full matrix. Even the TFIDF baseline is a bit higher, so we speculate that our preprocessing is the reason for the difference. In the end, our quadratic model surpasses the sparse model from their work by more than 10 MAP points.

On the cross-language dataset, the application of ℓ_1-regularization showed a completely different behavior: in this case, regularization is necessary and supports the algorithm to extract useful features. The final models we learned are relatively sparse, e.g. the best model we found in the experiments contains approximately $1,000,000$ unigram relations in a space of $50,000 \cdot 30,000 = 1,500,000,000$ possible pairs. This model is comparable with other unigram-based approaches: the Boost-1g (Sokolov et al., 2013) achieves a considerably higher PRES, but lower MAP score.

In our opinion, the unigram-based approach cannot be improved considerably more. We applied several filtering steps that resulted in statistically significant gains, however, the absolute increase is small. It would be interesting to extend the model with bigram-features, although the problem of increased model complexity remains unresolved at the moment.

We continued our investigation of the cross-language retrieval task on patents by creating a new model that incorporates special dense features defined on patent pairs. In addition to the domain knowledge features (Guo and Gomes, 2009), we used the score of the sparse model as a feature. This approach was extremely successful to further increase retrieval performance. As we use machine learning to train the model, a good approach is to define as many features as possible and let the machine learning application select useful weights.

The final dense model we examined shows strong performance on the patent retrieval task. According to MAP and PRES metric, our system scores highest in comparison to other SMT-based CLIR systems on the same dataset (Sokolov et al., 2013). Our system is even considerably better in MAP score than the best model based on bigrams, Boost-2g, but we are below by the PRES metric. The good results we achieved illustrates the usefulness of hand-crafted features for specialized tasks such as patent retrieval.

In a proof-of-concept experiment we showed that this model is able to easily incorporate other models. The idea of using multiple scores as dense features in combination with other highly informative (domain knowledge) features is extremely flexible and very powerful. The final model we examined in Section 5.3 is able to efficiently combine multiple CLIR systems and shows significant performance gains over each single system.

6.2 Future Work

Matrix Decomposition

One of the standard methods in information retrieval is singular value decomposition (SVD). The basic idea is to approximate a term-document matrix by one of lower rank. By cleverly constructing the matrix, the system is able to capture typical language phenomena like polysemy and synonymy. Actually, the matrix calculates similarity scores between query- and document-vectors in a lower dimensional "semantic" space, thus the name *latent semantic indexing (LSI)* or *latent semantic analysis (LSA)* for the application of SVD to text data.

The problem of LSI is its cost. For larger document collections, the initial matrix to calculate the SVD has high memory requirements (*vocabulary* $\times$ *documents*). Distributed computing can help, but then performance issues come up, because the computation itself is expensive. And finally, there is no way to optimize the similarity metric with respect to a certain information need, because LSI is an completely unsupervised task.

A different approach for matrix factorization was proposed by Koren et al. (2009). They describe how matrix factorization can be applied to the problem of recommender systems. Here, the problem is to find recommendations for a user based on his previous preferences. This is accomplished by projecting both user and target items into a lower dimensional latent space. The difference to LSI is that this approach does not use SVD to calculate the projection, but it can learn from a supervised signal that contains previous selections made by the user.

Bai et al. (2010) transfer this idea to retrieval and call it *supervised semantic indexing (SSI)*. Their approach is to approximate the matrix M used in a model similar to our basic model by the following equation:

$$\mathbf{M} = \mathbf{U}^\intercal \mathbf{V} \tag{6.1}$$

Here, $\mathbf{U}$ and $\mathbf{V}$ are $N \times \mathcal{D}$ matrices. Putting this decomposed matrix into the basic model actually means to project $\mathbf{q}$ and $\mathbf{d}$ into a latent space of dimension N and calculate a dot-product there:

$$\mathbf{q}^\intercal(\mathbf{U}^\intercal\mathbf{V})\mathbf{d} = (\mathbf{q}^\intercal\mathbf{U}^\intercal)(\mathbf{V}\mathbf{d}) = (\mathbf{U}\mathbf{q})^\intercal(\mathbf{V}\mathbf{d})$$

This illustrates that the two matrices of the decomposition, $\mathbf{U}$ and $\mathbf{V}$, project query and document independently. Furthermore, the matrices $\mathbf{U}$ and $\mathbf{V}$ can have different number of columns (but not rows), which makes them ideal for the application in a cross-language setup where the vocabulary sizes are usually uneven.

The independence of the matrix components enables a handy update formula for pairwise ranking that is based on the SGD strategy and modifies the components in sequence (Bai et al., 2010):

$$\left.\begin{array}{rcl} \mathbf{U} & \leftarrow & \mathbf{U} + \lambda\mathbf{V}(\mathbf{d}^+ - \mathbf{d}^-)\mathbf{q}^\intercal \\ \mathbf{V} & \leftarrow & \mathbf{V} + \lambda\mathbf{V}\mathbf{q}(\mathbf{d}^+ - \mathbf{d}^-)^\intercal \end{array}\right\} \text{ if } b_i - f(\mathbf{q}, \mathbf{d}^+) + f(\mathbf{q}, \mathbf{d}^-) > 0$$

Bai et al. (2010) have successfully tested this idea on a cross-language task based on Wikipedia article retrieval. In fact, the learning tool we used, Vowpal Wabbit, has a mode for matrix decomposition that follows the idea of the recommender systems approach by Koren et al. (2009). However, quick experiments under this configuration did not return useful results. The reason might be the differences in the implementation of the decomposed matrix, because the recommender systems are trained on sparse vectors, while the vectors for the basic model of our IR system have many features. Still, it is a very interesting approach and we would like to examine this further.

Latent Dirichlet Allocation

Another modern and very successful approach for dimension reduction is latent Dirichlet allocation (LDA) first described by Blei et al. (2003). Methods based on LDA became very popular in recent years. LDA is a generative hierarchical Bayesian model where documents are modeled as mixtures of underlying topics. These topics are again mixtures of topic probabilities.

The intuition behind LDA is that the following generative process creates the documents $\mathbf{w}$ in a corpus D (Blei et al., 2003):

1. Choose $N \sim \text{Poisson}(\xi)$
2. Choose $q \sim \text{Dir}(\alpha)$.
3. For each of the N words w_n:
   a Choose a topic $z_n \sim \text{Multinomial}(\theta)$.
   b Choose a word w_n from $p(w_n|z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

Here, N is a normal distribution, θ is the multinomial distribution over topics, and α and β are hyperparameters that control the sparsity of the topic distribution. Assuming this model, the idea is to induce these topic probabilities to the document. Methods such as Expectation Maximization (EM), Variational Bayes and Monte-Carlo-Markov-Chain (MCMC) have proven to be efficient methods to estimate the latent multinomial variables in an unsupervised setup.

This, in turn, gives the topic probabilities for a document which provides an explicit representation of the document. Our idea is to use this low dimensional document representation, actually a mixture of topics, to learn a relevance matrix as in the basic model in Section 2.3. The fact that we do not know which topics we obtain is in fact not relevant, because as long as we learn the matrix, any possible pairing is considered by the model.

One positive aspect of this model is that it would be very fast to train as soon as the low-dimensional vectors are created. Typically the latent dimensions are only a few hundred in numbers, thus the resulting model is very small. However, the inflexibility of the LDA

model is a disadvantage. Changing the corpus, i.e. by adding documents, or using an unseen document as query, makes it theoretically necessary to recreate the topics and re-learn the whole model.

Heinrich (2008) describes a method based on Gibbs sampling that is able to induce topic distributions on new documents without a rerun of the topic-creation process. They emphasize that this is an approximative solution, so after substantial changes to the corpus, the performance will al well degrade and it is necessary to recreate the topics and retrain the model.

Distributed Algorithms

The system we used for the sparse model was trained on a dataset of approximately 1 million documents (see Table 5.1). These documents can be combined for training in several ways, and we found out that 5 million training instances already return to good results. However, a dataset like MAREC (see Section 5.1) contains 19 million documents, which means to increase the training complexity by a factor of approximately 20. For the sparse models we trained, this would mean training times of several days per model. More data helps when complexity increases, so there is no end to the demand in computational power of such systems.

Agarwal et al. (2011) show how to build a "terascale linear learning system" that is aimed at optimizing a problem similar to the optimization problem defined by our basic model. The contribution of their paper is the implementation of an AllReduce-operation in the Hadoop/MapReduce-framework, that follows the communication scheme of the function with the same name in the Message Passing Interface framework (MPI). Their AllReduce-operation, however, is more sophisticated in terms of fault-tolerance and can automatically average local gradients with respect to a "confidence" value for each feature.

A similar approach based on a different task but still in a parallel computing setup is multitask-learning. Simianer et al. (2012) presented such a method applied to SMT and trained different models on different partitions of data to combine the partial results in an intelligent manner that also promotes useful features. They call their method multitask-learning with ℓ_1/ℓ_2-regularization and they prove the efficiency of this approach in a machine translation task.

These ideas can in principle be applied to our learning setup. In multitask-learning, the strategy is to train systems on different tasks and then combine these systems in a manner that increases overall system quality. The question in our setup is how to define tasks that are suitable for this type of learning. It could be beneficial to use patents from certain sections or classes, or grouped by years or origin.

The paper that first described ℓ_1- and ℓ_2-regularization for joint covariate selection was published by Obozinski et al. (2010). In this article the authors describe a intriguing approach that is able to automatically tune the ℓ_1/ℓ_2-regularization parameters. Judging from the strong influence of this parameter in our work (see e.g. figures 5.8 and 5.9) and the expensive parameter search, this is a fascinating idea that could be helpful in many different

setups.

Thus, these two aspects, distributed learning and automatic parameter estimation, are two directions we would like to follow in the future.

Fast-Response Retrieval with MPI

The current implementation is slow compared to other retrieval systems. As we have already mentioned in Section 3.1, the focus was not to implement a fast running system but to create a flexible model where different configurations can be exchanged and tested easily.

Evaluations of our dense model are expensive. It is required to have the sparse model, the database and the dense model simultaneously running. Testing 2000 query patents against a collection of 100,000 patents takes about 20 minutes on our 30-node Hadoop-cluster (with 240 cores), which are about 167,000 checked pairs per second. This means if a user sends a query to a patent database like MAREC with 19 million patents, it would take approximately 2 minutes to process the query until he will be presented a single result.

In addition to this, Hadoop is a system that is not optimized for retrieval tasks. In fact, it is completely unsuitable for fast-response systems, because the overhead of starting a job in the MapReduce-framework alone consumes tens of seconds and can be even longer if other Hadoop jobs are running in parallel. The application of evaluation is not problematic regarding this delay, because we are evaluating many queries against many documents, and the delay until we have the result is less relevant.

For fast-response systems, a low latency is crucial. It is thus advisable to move away from the Hadoop-framework to a distributed system that has lower latency by design. We now describe a distributed system that is based on MPI. This systems addresses two problems of the current setup:

1. High latency due to Hadoop-job setup.
2. Slow document retrieval on large corpora.

Even by increasing the number of Hadoop-cluster nodes, the overhead of starting a Hadoop-process does not change. This latency in the order of tens of seconds up to a minute is a natural barrier for Hadoop-based jobs and this makes such systems prohibitive for normal retrieval task, where the user wants feedback as fast as possible.

Thus, the only way to change this problem is to move from a Hadoop-based architecture to a distributed system that was built with low-latency technology in mind: we think of a distributed memory systems based on MPI. The largest MPI-nodes available to us have 40 cores per machine. As these are more powerful than the Hadoop-nodes, we optimistically assume 80,000 patent evaluations per second. To achieve sub-second response time, we would need in the best case 237 such machines ($19 \cdot 10^6/80,000 = 237.5$), and this is still far to optimistic, because we had not considered the necessary messaging overhead to broadcast the query to all nodes and to reduce the partial results.

Our solution is to use the fact that the basic model is linear in each of its arguments. We can thus employ k-means clustering, an unsupervised method for grouping data, as a pre-processing step generating artificial "representers", i.e. the centroids, for each cluster group.

Moving from one cluster to another along a line in the space where our documents are embedded changes the score linearly. Given a query $\mathbf{q}$, we can start by generating k scores against the k-centroids of each cluster. The two best representers, i.e. the two centroids that return the highest relevance score for $\mathbf{q}$, denote the two clusters where the documents of interest are located. Figure 6.1 illustrates the idea: 1) We evaluate a query $\mathbf{q}$ against k artificial data points. 2) We select the top-2 clusters by highest retrieval score. 3) Now, we only need to evaluate our query against the documents in these two clusters to find the best scoring documents.

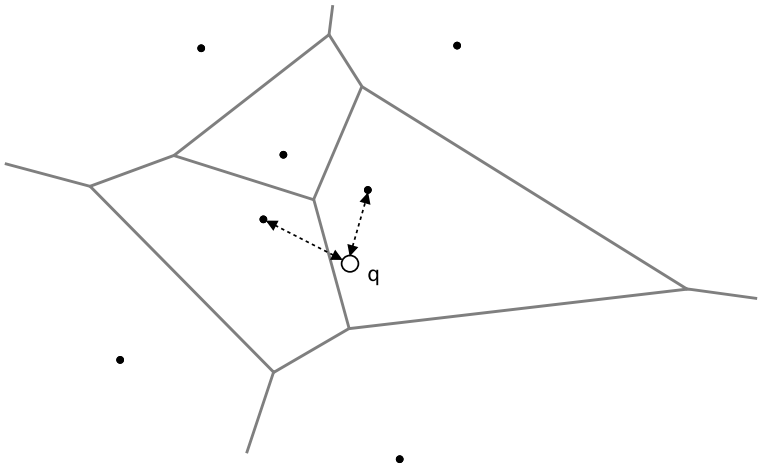

Figure 6.1: Voronoi-diagram showing clusters with their centroids and a query q in document space.

This way we can considerably reduce the number of calculations by several orders of magnitude. For example, assuming clusters with 10,000 elements on average, we will have 1,900 cluster centroids. Our system has to evaluate first the query against 1,900 centroids, then sort the results and select the top-2 clusters. These cluster contain $2 \times 10,0000$ documents, thus the total number of documents to evaluate drops to roughly $22,000$. With the assumption that our MPI-node can execute $80,000$ evaluations per second, we are well below the targeted one-second latency.

However, this sounds too good to be true, and indeed there are several aspects we left out. First, we need to calculate the centroids of 19 million data points in a space that exceeds 350,000 dimensions (see e.g. Table 5.1 on vocabulary size) – this is a task of its own dimension. Second, we need to get the relevant parts of the patent data into memory. The MAREC corpus is too large to load it into memory on a single machine (approx. 620 GB text files), so it is necessary to distribute the documents on several machines and use MPI

to send the query and aggregate the results through messages. A third aspect is that there is no guarantee that we find the optimal solution by looking only at the members of the two nearest clusters. Assuming the data points are well distributed, the probability of misses is very low, but these bounds have to be evaluated.

The first two issues are technically solvable, but there is one additional point that prohibits the use of this method in combination with our best dense model: the idea behind k-means construction for pre-selection depends on the properties of the *basic* model, which we implemented in our sparse model. The dense model, however, uses a different approach and the required linearity in arguments does not hold there.

Integrating Other Systems

We have already proven the extensibility of our model in Section 5.3 by successfully incorporating the Boost-2g model into our dense model. Viewing the dense model as one layer of a multi-layer learner almost advises to add more models to this system.

Highly probable candidates for integration as a module into the dense model come from Hieber et al. (2013) and Jehl et al. (2012). Both follow more classical CLIR approaches based on translations of query terms and full query sentences. For combination of different models, a high diversity of approaches is beneficial. If all modules more or less generate the same ranking, than the dense model cannot improve anything. However, if the models follow different approaches, the dense model can find the optimal combination.

Sokolov et al. (2013) presented a system where their SMT-based model is combined with the Boost-2g model using an unsupervised method called Borda counts. We can simply continue to improve this result by integrating these two systems as a feature into our dense model. It would be very interesting to examine how the addition of the other SMT-based systems can further improve the final retrieval performance.

However, affordable computer resources are limited. Judging from our experience with the model, we believe we can integrate several modules and still run evaluation tests in sub-hour time on the Hadoop-cluster. If we need more than that, there is still the possibility to move to MPI, which is in the medium run a good choice for the retrieval part of such systems and due to its scalability not at all a bad choice for training systems of growing complexity.

Chapter 7

Conclusion

We presented a retrieval system based on a learning to rank approach that goes beyond the weighting of single terms by taking all possible word-pairs into account. Such systems naturally have high demands in memory and computational complexity, because the parameter space they work on is quadratic in the size of the vocabulary.

Several methods that aim at controlling the resource usage were examined in this work. To solve the optimization problem efficiently, we used a stochastic gradient approach that can theoretically work on an unlimited number of training points. This has become a standard technique especially in applications where the number of parameters is very high and contains noise, as it is often the case in natural language processing tasks.

We implemented a tool that created static training data on disk using the combination of the stochastic approach and pairwise ranking, so we were able to compare different settings of the learner on the same data set. A very sophisticated and resource conserving feature of our learning tools is the on-line expansion of quadratic vectors. This reduces the required disk space by several orders of magnitude, makes better use of the I/O-performance and decreases the time needed to train the models significantly.

Regarding feature space reduction, we compared several methods that aim directly at reducing the parameter space. Hashing the weight vector has proven to be a very effective method to reduce the memory requirements of a system down to a small fraction of the initial requirements. However, hashing can affect the learning process negatively if the problem of collisions is not adequately considered.

An efficient way to employ hashing and reduce collisions is to use correlated feature hashing with multiple bins, where multiple hash functions are constructed that hash similar words to similar buckets. The way these mappings are constructed is completely different to other approaches such as locality sensitive hashing. The result and the final retrieval quality we observed proves that the method is suitable to reduce feature space on one hand, and increase retrieval quality on the other hand.

While moving to the cross-language system, we found out that not all settings that worked well on the monolingual setting can be applied to the cross-language task. To our surprise, the TFIDF-vectors were not suited for the task of learning relevance and word

relations simultaneously. Another difference is that ℓ_1-regularization did not work well in the monolingual setup, but it was absolutely necessary in the cross-language task to achieve a descent performance.

The monolingual and the cross-language task reacted differently to the limiting of the dictionary for feature space reduction. While in the monolingual setup the performance degraded significantly for the quadratic model, there were practically no differences in the cross-language quadratic model's case. Nevertheless, we did not find a conclusive explanation for this. After all, the monolingual model behaved not exactly as expected in many respects, which we attribute to the data and the task. However, we achieved unexpectedly high retrieval performance with the monolingual quadratic model.

Besides methods that had only very little effect in the cross-language setting, we also employed several other reduction methods which increased the retrieval performance considerably. By carefully analyzing each contribution it was gratifying to find out that the improvement of the single methods, each of them changing the previous system significantly, all added up in the end to the final result. The method of correlated feature hashing showed a strong positive influence to the retrieval performance for both tasks and is thus a method that we can definitely recommend, especially in combination with systems that work with hashed parameter spaces due to their high dimensionality.

One of the initial questions was that if we can learn a relevance metric and at the same time learn bilingual correlations of terms from a supervised signal based on cross-language retrieval. Our final experiments on the sparse model achieved ranking performance on par with other unigram-based systems and this proves that cross-language retrieval must not necessarily contain a module for machine translation. We have further shown that a relatively simple machine learning approach is able to capture complex word-relations across languages.

The probably most surprising result is the success of the model which integrated the sparse model into the dense model. To our knowledge, a comparable approach to combine highly informative features with sparse features from a completely different model has not been done before. We have empirically shown that the model is able to integrate a (theoretically) arbitrary number of ranking modules optimally, because the best combination is learned using machine learning techniques on the full ranking. This is the difference to supervised rank aggregation, and it emphasizes that our model blurs the line which is drawn between rank creation and rank aggregation.

Bibliography

Agarwal, Alekh, O. Chapelle, M. Dudík and J. Langford (2011). *A Reliable Effective Terascale Linear Learning System*. CoRR, abs/1110.4198.

Aslam, Javed A. and M. H. Montague (2001). *Models for Metasearch*. In Croft, W. Bruce, D. J. Harper, D. H. Kraft and J. Zobel, eds.: *SIGIR*, pp. 275–284. ACM.

Bai, Bing, J. Weston, D. Grangier, R. Collobert, K. Sadamasa, Y. Qi, O. Chapelle and K. Q. Weinberger (2010). *Learning to rank with (a lot of) word features*. Inf. Retr., 13(3):291–314.

Barroso, Luiz André and U. Hölzle (2009). *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers.

Bashir, Shariq and A. Rauber (2010). *Improving Retrievability of Patents in Prior-Art Search*. In Gurrin, Cathal, Y. He, G. Kazai, U. Kruschwitz, S. Little, T. Roelleke, S. M. Rüger and K. van Rijsbergen, eds.: *ECIR*, vol. 5993 of *Lecture Notes in Computer Science*, pp. 457–470. Springer.

Bi, Jinbo, K. P. Bennett, M. J. Embrechts, C. M. Breneman and M. Song (2003). *Dimensionality Reduction via Sparse Support Vector Machines*. Journal of Machine Learning Research, 3:1229–1243.

Blei, David M., A. Y. Ng, M. I. Jordan and J. Lafferty (2003). *Latent Dirichlet allocation*. Journal of Machine Learning Research, 3:2003.

Blum, Avrim, A. Kalai and J. Langford (1999). *Beating the Hold-Out: Bounds for K-fold and Progressive Cross-Validation*. In Ben-David, Shai and P. M. Long, eds.: *COLT*, pp. 203–208. ACM.

Bottou, Léon and O. Bousquet (2008). *The Tradeoffs of Large Scale Learning*. In Platt, J.C., D. Koller, Y. Singer and S. Roweis, eds.: *Advances in Neural Information Processing Systems*, vol. 20, pp. 161–168. NIPS Foundation (http://books.nips.cc).

Boyd, Stephen and L. Vandenberghe (2004). *Convex Optimization*. Cambridge University Press.

Burges, Christopher J. C., R. Ragno and Q. V. Le (2006). *Learning to Rank with Nonsmooth Cost Functions*. In Schölkopf, Bernhard, J. Platt and T. Hoffman, eds.: *NIPS*, pp. 193–200. MIT Press.

Chapelle, Olivier (2007). *Training a Support Vector Machine in the Primal*. Neural Computation, 19(5):1155–1178.

Chapelle, Olivier and Y. Chang (2011). *Yahoo! Learning to Rank Challenge Overview*. Journal of Machine Learning Research - Proceedings Track, 14:1–24.

Charikar, Moses (2002). *Similarity Estimation Techniques from Rounding Algorithms*. In *STOC*, pp. 380–388.

Chen, Xi, B. Bai, Y. Qi, Q. Lin and J. G. Carbonell (2010). *Learning Preferences with Millions of Parameters by Enforcing Sparsity*. In Webb, Geoffrey I., B. L. 0001, C. Zhang, D. Gunopulos and X. Wu, eds.: *ICDM*, pp. 779–784. IEEE Computer Society.

Cohen, Paul R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge.

Collobert, Ronan and S. Bengio (2004). *Links between perceptrons, MLPs and SVMs*. In Brodley, Carla E., ed.: *ICML*, vol. 69 of *ACM International Conference Proceeding Series*. ACM.

Cortes, Corinna, M. Mohri and A. Rastogi (2007). *Magnitude-preserving ranking algorithms*. In Ghahramani, Zoubin, ed.: *ICML*, vol. 227 of *ACM International Conference Proceeding Series*, pp. 169–176. ACM.

Cortes, Corinna and V. N. Vapnik (1995). *Support Vector Networks*. Machine Learning, 20:273–297.

Dean, Jeffrey and S. Ghemawat (2004). *MapReduce: Simplified Data Processing on Large Clusters*.

Deerwester, Scott, S. T. Dumais, G. W. Furnas, T. K. L and R. Harshman (1990). *Indexing by latent semantic analysis*. Journal of the American Society for Information Science, 41:391–407.

Dice, Lee Raymond (1945). *Measures of the Amount of Ecologic Association Between Species*. Ecology, 26(3):297–302.

Dyer, Chris, A. Lopez, J. Ganitkevitch, J. Weese, F. Ture, P. Blunsom, H. Setiawan, V. Eidelman and P. Resnik (2010). *cdec: A Decoder, Alignment, and Learning Framework for Finite-State and Context-Free Translation Models*. In *Proceedings of the Association for Computational Linguistics (ACL)*.

Fendrich, Sascha (2012). *Sol – Stochastic Learning Toolkit*. Technical Report, Department of Computational Linguistics, Heidelberg University.

Flynn, Michael J. (1972). *Some Computer Organizations and Their Effectiveness*. Computers, IEEE Transactions on, C-21(9):948–960.

Forman, George (2003). *An Extensive Empirical Study of Feature Selection Metrics for Text Classification*. Journal of Machine Learning Research, 3:1289–1305.

Friedman, Jerome H. (1999). *Greedy Function Approximation: A Gradient Boosting Machine*. Annals of Statistics, 29(5):1189–1232.

Gionis, Aristides, P. Indyk and R. Motwani (1999). *Similarity Search in High Dimensions via Hashing*. In *VLDB '99: Proceedings of the 25th International Conference on Very Large Data Bases*, pp. 518–529, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Graf, E. and L. Azzopardi (2008). *A methodology for building a patent test collection for prior art search*. In *The Second International Workshop on Evaluating Information Access (EVIA)*.

Guo, Yunsong and C. P. Gomes (2009). *Ranking Structured Documents: A Large Margin Based Approach for Patent Prior Art Search*. In Boutilier, Craig, ed.: *IJCAI*, pp. 1058–1064.

Guyon, I. and A. Elisseeff (2003). *An introduction to variable and feature selection*. The Journal of Machine Learning Research, 3:1157–1182.

Heaps, H. S. (1978). *Information Retrieval: Computational and Theoretical Aspects*. Academic Press, Inc., Orlando, FL, USA.

Heinrich, Gregor (2008). *Parameter estimation for text analysis*. Technical note version 2 (1: 2005), vsonix GmbH and University of Leipzig, http://www.arbylon.net/publications/text-est.pdf.

Hieber, Felix, L. Jehl and S. Riezler (2013). *Task Alternation in Parallel Sentence Retrieval for Twitter Translation*. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 323–327, Sofia, Bulgaria. Association for Computational Linguistics.

Hunter, John K. and B. Nachtergaele (2001). *Applied Analysis*. World Scientific Publishing Company Incorporated.

Indyk, Piotr and R. Motwani (1998). *Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality*. In *STOC*, pp. 604–613.

Jehl, Laura, F. Hieber and S. Riezler (2012). *Twitter translation using translation-based cross-lingual retrieval*. In *Proceedings of the Seventh Workshop on Statistical Machine Translation*, WMT '12, pp. 410–421, Stroudsburg, PA, USA. Association for Computational Linguistics.

Joachims, Thorsten (1997). *A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization*. In *Proceedings of the 14th International Conference on Machine Learning*.

Joachims, Thorsten (2002). *Optimizing search engines using clickthrough data*. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 133–142, New York, NY, USA. ACM.

Jones, Karen Spärck (1972). *A statistical interpretation of term specificity and its application in retrieval*. Journal of Documentation 28, pp. 11–21.

Jones, Karen Spärck, S. Walker and S. E. Robertson (2000). *A probabilistic model of information retrieval: development and comparative experiments - Part 2*. Information Processing and Management, 36(6):809–840.

Knuth, Donald E. (1973). *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley.

Knuth, Donald E., J. H. Morris and V. R. Pratt (1977). *Fast Pattern Matching in Strings*. SIAM Journal of Computing, 6(2):323–350.

Kononenko, Igor (1995). *On Biases in Estimating Multi-Valued Attributes*. In *IJCAI*, pp. 1034–1040. Morgan Kaufmann.

Kontorovich, Leonid (2007). *A Universal Kernel for Learning Regular Languages*. In *MLG*.

Koren, Yehuda, R. M. Bell and C. Volinsky (2009). *Matrix Factorization Techniques for Recommender Systems*. IEEE Computer, 42(8):30–37.

Kulis, Brian and K. Grauman (2012). *Kernelized Locality-Sensitive Hashing*. IEEE Trans. Pattern Anal. Mach. Intell., 34(6):1092–1104.

Lal, ThomasNavin, O. Chapelle, J. Weston and A. Elisseeff (2006). *Embedded Methods*. In Guyon, Isabelle, M. Nikravesh, S. Gunn and L. Zadeh, eds.: *Feature Extraction*, vol. 207 of *Studies in Fuzziness and Soft Computing*, pp. 137–165. Springer Berlin Heidelberg.

Levenshtein, Vladimir (1966). *Binary Codes Capable of Correcting Deletions and Insertions and Reversals*. Soviet Physics Doklady, 10(8):707–710.

Li, Hang (2011). *Learning to Rank for Information Retrieval and Natural Language Processing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Lin, Jimmy and C. Dyer (2010). *Data-Intensive Text Processing with MapReduce*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers.

Liu, Tie-Yan (2010). *Learning to Rank for Information Retrieval*. Springer-Verlag New York Inc.

Luxburg, Ulrike von and B. Schölkopf (2011). *Statistical Learning Theory: Models, Concepts, and Results*. In Gabbay, D. M., S. Hartmann and J. H. Woods, eds.: *Handbook of the History of Logic Vol. 10: Inductive Logic*, vol. 10, pp. 651 – 706. Elsevier North Holland, Amsterdam, Netherlands.

Magdy, Walid and G. J. F. Jones (2010). *PRES: a score metric for evaluating recall-oriented information retrieval applications*. In Crestani, Fabio, S. Marchand-Maillet, H.-H. Chen, E. N. Efthimiadis and J. Savoy, eds.: *SIGIR*, pp. 611–618. ACM.

Manning, Christopher D., P. Raghavan and H. Schuetze (2008). *Introduction to Information Retrieval*. Cambridge University Press.

Melamed, I. Dan (1998). *Models of Co-occurrence*. CoRR, cmp-lg/9805003.

Message Passing Interface Forum (2009). *MPI: A Message-Passing Interface Standard, Version 2.2*. Specification.

Metropolis, Nicholas and S. Ulam (1949). *The Monte Carlo method*. J. Am. Stat. Assoc., 44:335.

Murphy, Kevin P. (2012). *Machine Learning: A Probabilistic Perspective (Adaptive Computation and Machine Learning series)*. The MIT Press.

Narlikar, Girija J. and G. E. Blelloch (1998). *Pthreads for Dynamic Parallelism*. Technical Report

Nocedal, Jorge and S. J. Wright (2006). *Numerical optimization*.

Noreen, Eric W. (1989). *Computer-intensive methods for testing hypotheses*. A Wiley-Interscience publication. Wiley, New York, NY [u.a.].

Obozinski, Guillaume, B. Taskar and M. I. Jordan (2010). *Joint covariate selection and joint subspace selection for multiple classification problems*. Statistics and Computing, 20(2):231–252.

Ojala, Markus and G. C. Garriga (2010). *Permutation Tests for Studying Classifier Performance*. Journal of Machine Learning Research, 11:1833–1863.

OpenMP Architecture Review Board (2008). *OpenMP Application Program Interface Version 3.0*.

Page, Lawrence, S. Brin, R. Motwani and T. Winograd (1999). *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report, Stanford University.

Peters, Carol, M. Braschler and P. Clough (2012). *Multilingual Information Retrieval - From Research To Practice*. Springer.

Peters, Carol, G. M. D. Nunzio, M. Kurimo, T. Mandl, D. Mostefa, A. Peñas and G. Roda, eds. (2010). *Multilingual Information Access Evaluation I. Text Retrieval Experiments, 10th Workshop of the Cross-Language Evaluation Forum, CLEF 2009, Corfu, Greece, September 30 - October 2, 2009, Revised Selected Papers*, vol. 6241 of *Lecture Notes in Computer Science*. Springer.

Piroi, Florina, M. Lupu, A. Hanbury and V. Zenz (2011). *CLEF-IP 2011: Retrieval in the Intellectual Property Domain*. In Petras, Vivien, P. Forner and P. D. Clough, eds.: *CLEF (Notebook Papers/Labs/Workshop)*.

Porter, Martin F. (1980). *An Algorithm for Suffix Stripping*. Program, 14(3):130–137.

Radivojac, Predrag, Z. Obradovic, A. K. Dunker and S. Vucetic (2004). *Feature Selection Filters Based on the Permutation Test*. In Boulicaut, Jean-François, F. Esposito, F. Giannotti and D. Pedreschi, eds.: *ECML*, vol. 3201 of *Lecture Notes in Computer Science*, pp. 334–346. Springer.

Rahimi, Ali and B. Recht (2007). *Random Features for Large-Scale Kernel Machines*. In Platt, John C., D. Koller, Y. Singer and S. T. Roweis, eds.: *NIPS*. Curran Associates, Inc.

Riezler, Stefan and J. T. Maxwell (2005). *On Some Pitfalls in Automatic Evaluation and Significance Testing for MT*. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pp. 57–64, Ann Arbor, Michigan. Association for Computational Linguistics.

Roberts, Larry (1986). *The Arpanet and computer networks*. In *Proceedings of the ACM Conference on The history of personal workstations*, HPW '86, pp. 51–58, New York, NY, USA. ACM.

Rosenblatt, Frank (1958). *The perceptron: a probabilistic model for information storage and organization in the brain*. Psychological Reviews, 65(6):386–408.

Ruppert, Eugen (2013). *Cross-Lingual Patent Retrieval for Large Data Collections*. Master's thesis, Ruprecht-Karls-Universität Heidelberg, Heidelberg, Germany.

Salton, Gerard, A. Wong and C.-S. Yang (1975). *A Vector Space Model for Automatic Indexing*. Communications of the ACM, 18(11):613—620.

Sarasúa, Leo and G. Corremans (2000). *Cross Lingual Issues in Patent Retrieval*.

Schölkopf, Bernhard and A. J. Smola (2002). *Learning with Kernels*. MIT Press.

Sculley, D. (2009). *Large Scale Learning to Rank*. In *NIPS 2009 Workshop on Advances in Ranking*.

Shakhnarovich, Gregory, P. A. Viola and T. Darrell (2003). *Fast Pose Estimation with Parameter-Sensitive Hashing*. In *ICCV*, pp. 750–759. IEEE Computer Society.

Shen, Libin and A. K. Joshi (2005). *Ranking and Reranking with Perceptron*. Machine Learning, 60(1-3):73–96.

Shi, Qinfeng, J. Petterson, G. Dror, J. Langford, A. Smola and S. Vishwanathan (2009). *Hash Kernels for Structured Data*. J. Mach. Learn. Res., 10:2615–2637.

Simianer, Patrick, S. Riezler and C. Dyer (2012). *Joint Feature Selection in Distributed Stochastic Learning for Large-Scale Discriminative Training in SMT*. In *ACL (1)*, pp. 11–21. The Association for Computer Linguistics.

Singhal, Amit (2001). *Modern Information Retrieval: A Brief Overview*. IEEE Data Eng. Bull., 24(4):35–43.

Smadja, Frank, K. R. McKeown and V. Hatzivassiloglou (1996). *Translating Collocations for Bilingual Lexicons: A Statistical Approach*. Computational Linguistics, 22(1):1–38.

Smucker, Mark D., J. Allan and B. Carterette (2007). *A comparison of statistical significance tests for information retrieval evaluation*. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pp. 623–632, New York, NY, USA. ACM.

Sokolov, Artem, L. Jehl, F. Hieber and S. Riezler (2013). *Boosting Cross-Language Retrieval by Learning Bilingual Phrase Associations from Relevance Rankings*. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Tibshirani, Robert (1996). *Regression Shrinkage and Selection via the Lasso*. Journal of the Royal Statistical Society (Series B), 58:267–288.

Tsuruoka, Yoshimasa, J. Tsujii and S. Ananiadou (2009). *Stochastic Gradient Descent Training for L1-regularized Log-linear Models with Cumulative Penalty*. In Su, Keh-Yih, J. Su and J. Wiebe, eds.: *ACL/IJCNLP*, pp. 477–485. The Association for Computer Linguistics.

Türe, Ferhan, J. J. Lin and D. W. Oard (2012). *Combining Statistical Translation Techniques for Cross-Language Information Retrieval*. In *COLING'12*, pp. 2685–2702.

Watkins, Chris (2000). *Dynamic Alignment Kernels*. In Smola, A. and P. Bartlett, eds.: *Advances in Large Margin Classifiers*, ch. 3, pp. 39–50. MIT Press, Cambridge, MA, USA.

Weinberger, Kilian Q., A. Dasgupta, J. Attenberg, J. Langford and A. J. Smola (2009). *Feature Hashing for Large Scale Multitask Learning*. CoRR, abs/0902.2206.

Weston, Jason, A. Elisseeff, B. Schölkopf and M. E. Tipping (2003). *Use of the Zero-Norm with Linear Models and Kernel Methods*. Journal of Machine Learning Research, 3:1439–1461.

Wu, Qiang, C. J. Burges, K. M. Svore and J. Gao (2008). *Ranking, Boosting, and Model Adaptation*.

Yu, Yuan, M. Isard, D. Fetterly, M. Budiu, U. Erlingsson, P. K. Gunda and J. Currey (2008). *DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language*. In Draves, Richard and R. van Renesse, eds.: *OSDI*, pp. 1–14. USENIX Association.

Declaration

I do hereby solemnly declare that I have completed the preceding *Thesis for the Degree of Master of Arts in Computational Linguistics* independently, and have not used any other sources or aids apart from those listed.

Heidelberg, 26.09.2013 Schigehiko Schamoni