Universidade Federal do Amazonas
Instituto de Computação
Programa de Pós-Graduação em Informática

Aplicação de Técnicas de Aprendizagem de Máquina na Geração de Índices para Sistemas de Busca

André Luiz da Costa Carvalho

Manaus – Amazonas
Novembro de 2012

André Luiz da Costa Carvalho

Aplicação de Técnicas de Aprendizagem de Máquina na Geração de Índices para Sistemas de Busca

Tese apresentada ao Programa de Pós Graduação em Informática da Universidade Federal do Amazonas, como requisito parcial para a obtenção do título de Doutor em Informática, área de concentração em Banco de Dados e Recuperação de Informação.

Advisor: Prof. Edleno Silva de Moura, Ph.D.

André Luiz da Costa Carvalho

Aplicação de Técnicas de Aprendizagem de Máquina na Geração de Índices para Sistemas de Busca

Tese apresentada ao Programa de Pós Graduação em Informática da Universidade Federal do Amazonas, como requisito parcial para a obtenção do título de Doutor em Informática, área de concentração em Banco de Dados e Recuperação de Informação.

Manaus – Amazonas

Novembro de 2012

O cientista não é o homem que fornece as verdadeiras respostas;

é quem faz as verdadeiras perguntas.

Claude Levi Strauss.

Resumo

Máquinas de busca estão entre as principais formas de se obter informações na internet, recebendo milhões de consultas diárias. Este volume avantajado de consultas gera uma considerável carga nos processadores de consultas das máquinas de busca, que devem não apenas se preocupar com a qualidade da resposta final recebida pelos usuários, mas também com a latência desta consulta, já que uma demora excessiva no tempo de resposta pode prejudicar a experiência de seus usuários.

Nos últimos anos tem havido um considerável esforço de pesquisa na aplicação de técnicas de aprendizado de máquina durante o processamento de consultas, objetivando-se principalmente um aumento na qualidade final de suas respostas. Nesta tese estudamos a aplicação de técnicas de aprendizagem de máquina durante a geração de índices, ao invés de aplicá-las ao processamento de consultas, abrindo portanto uma nova frente para a aplicação de técnicas de aprendizagem de máquina em sistemas de busca. Dentro do estudo, propomos duas técnicas para a aplicação de aprendizado de máquina na indexação de documentos em máquinas de busca, mostrando com isso que há espaço para melhorar a indexação com o uso dessas técnicas.

A vantagem de tal abordagem é que, como esse processamento é feito antes das consultas serem feitas à maquina de busca, independente de quão custoso computacionalmente seja este processo, isto não refletirá diretamente no tempo de processamento de consultas. Propomos aqui duas técnicas: LePrEF, uma técnica de fusão de evidências em tempo de indexação que tem como objetivo a melhoria do desempenho de máquinas de busca durante o processamento de consultas, por meio da geração de índices que codificam melhor a importância de cada termo em cada documento, e uma técnica de detecção de termos frasais (os sintagmas), com o objetivo de melhorar a qualidade das respostas obtidas por elas.

A técnica LePrEF realiza a fusão de fontes de evidência de relevância em tempo de indexação utilizando para tanto aprendizagem de máquina. A necessidade da fusão de evidências no processamento de consultas deriva do fato de que sistemas de busca em geral utilizam diversas fontes de evidência para computar suas respostas, tais como o texto das páginas web, o texto dos apontadores recebidos por cada página, métodos de análise de apontadores como o PageRank, dentre muitos outros. Porém, o acréscimo de novas fontes de evidência leva também a novos custos de processamento. Isto, aliado ao constante crescimento na quantidade de conteúdo presente na Web, leva a um sempre crescente aumento na carga de processamento feita pelos processadores de consultas. Desta forma, qualquer abordagem que leve à uma melhoria na eficiência no processamento de consultas pode ser considerada essencial, bem como estratégias que levem a um aumento na qualidade sem contudo levar a aumento nos custos de processamento.

Uma das principais desvantagens do uso de múltiplas fontes de evidência é que, nas abordagens tradicionais, cada uma delas é processada individualmente, para depois suas informações serem fundidas em um único valor durante o processamento das consultas. Apesar da abordagem levar a uma qualidade satisfatória, ela também leva a um grande aumento no tempo de processamento. Nesta tese propomos uma abordagem para fazer a fusão de evidências durante a indexação, diminuindo a carga extra adicionada pelas múltiplas evidências no processamento de consultas.

Nesta tese também é proposto um novo método para adicionar bigramas selecionados ao índice com objetivo de melhorar a qualidade das respostas às consultas, sem contudo levar a um aumento no custo de processamento. Para tanto, propusemos um método que utiliza aprendizagem de máquina para a detecção automática de termos frasais, que são pares de termos que, quando co-ocorrendo, possuem um significado específico diverso dos termos que o formam.

Avaliamos também o impacto que o uso em conjunto das duas técnicas propostas teria sobre um sistema de busca, avaliando o impacto que estas duas abordagens podem ter não somente na eficiência do processamento de consultas, mas também na sua qualidade de resposta. Os resultamos obtidos indicam que a nova frente para o uso de técnicas de aprendizagem de máquina em sistemas de busca é bastante promissora.
Palavras Chaves: Aprendizado de Máquina, Maquina de Busca, Indexação

Contents

List of Figures

List of Tables

Aplicação de Técnicas de Aprendizagem de Máquina na Geração de Índices para Sistemas de Busca

December 20, 2012

Abstract

Search Engines are among the main methods of obtaining information on the Web, receiving millions of queries per day. Not only is this volume large, but also the number of documents present in their datasets is of a very large order of magnitude, in the order of billions of documents. These search engines often use a number of data indexing techniques in order to not only achieve high quality answers, but also achieve them with as little a computational cost as possible.

Recently, there has been an ongoing effort in applying machine learning techniques in query processing, usually aiming at enhancing the quality of the results. In this thesis, we study two applications of machine learning techniques not at the query processing stages, but at **indexing time**.

In particular, we propose two techniques to introduce machine learning in the indexing stages of search engines: LePrEF, a learning to rank technique that fuses sources of relevance evidence during indexing time, aiming at reducing the query processing load by performing evidence fusion during indexing time; and a technique to detect pairs of terms that, when present side-by-side, compound an specific meaning, which we call in this thesis phrasal terms. We then use these phrasal terms to enhance the document representation present in the index of search engines, with the objective of improving the final result answer.

We also evaluated the impact of applying both techniques to a search system. Experimental results indicate that there is a synergy with them, leading to the best overall results present in our experiments. These results are a strong indication that the use of machine learning to enhance the indexing of search engine datasets is indeed very promising, and a possible strong research venue.

Chapter 1

Introduction

Nowadays, search engines represent the main tool to locate information on the Web, receiving millions of daily queries. For instance, comScore[1] measured that, when considering only U.S. users, Google Search received 11.3 billion queries in August of 2012, an average of about 364 million daily queries.

Two factors stand out prominently regarding the satisfaction of search engine users (Al-Maskari and Sanderson, 2010): the **quality** of search results and the **time** needed to fulfill it. While initially it could be arguable that all that matters is the final quality of the results, users would probably replace a search engine that yields good answers but with an elevated latency for one that has slightly worse results but remarkably faster. Further, more efficient systems may also reduce the costs for processing queries, which is also an important factor for a search engine company.

Besides the large volume of received queries, another important factor in search tasks is the growing size of their datasets. The Web Site `www.worldwidewebsize.com` has estimated

[1]www.comscore.com

that in October 2012 Google[2] had about 40 **billion** pages present in its index. Due to this extremely large number of pages, even the most simple tasks, when applied to the whole dataset, can lead to potentially large computational costs if not properly optimized. Not only that, but this abundance of information may also introduce new challenges when trying to differentiate documents that satisfy users needs from the ones that do not.

This massive growth of the dataset size, added to the large number of queries daily submitted to modern search engines, constitute a strong evidence that efficiency is one of the major concerns when designing search engines, besides the quality of the answer provided to its users.

Regarding quality issues, a very popular approach for processing queries on search engines is to adopt Machine Learning methods. These methods can be defined as a body of techniques whose main objective is to learn patterns present in data provided as input (Baeza-Yates and Ribeiro-Neto, 1999). In case of search engines, machine learning techniques are usually adopted as techniques for learning to rank documents given a query. This learning process consists in taking examples showing how the documents should be ranked given a query, which is known as training data, and then derive ranking methods that can be applied to new queries.

One of the main advantages of using Machine Learning methods is that you can, to some extent, affirm that those methods are data collection agnostic, i.e., since the method is reliant on patterns present in the training data, it will, a priori, be adapted to particularities of the scenario where it is being applied.

In this thesis, we are interested in applying learning techniques at the indexing stage of search engine database construction. We present two machine learning applications to enhance the indexing phase of search engines, expecting to enhance the computational efficiency and final quality of the results obtained. One approach is based on using Machine Learning to perform

[2]www.google.com

evidence fusion at indexing time, and the other is to use machine learning to detect, among all the bigrams that co-occur in text documents, those which have a specific meaning, in order to use them to enhance search quality. We also present how both methods synergize with each other, leading to a broader vision on how the contributions present in this thesis can push forward search technology.

Our technique to perform evidence fusion at indexing time can be seen as a new indexing paradigm. Search systems usually adopt several distinct sources of relevance evidence to compute their final ranking. Examples of these sources of evidence are: Web link graph analysis, the use of the anchor text of the received links of a Web page, their HTML structure, such as titles and headings, URL tokenization, among many others.

The use of these sources and new ones that are continually proposed in literature leads to a problem: how to leverage the use of these different sources in harmony, in such a way that the best possible answer ranking is obtained for each query. Learning to rank techniques are among the most successful approaches to solve this problem. Examples of such techniques are SVM Rank (Joachims, 2002), RankBoost (Freund et al., 2003) and Genetic Programming based methods (de Almeida et al., 2007, Silva et al., 2009).

Learning to rank techniques are adopted in literature to fuse evidences at query processing time. However, the addition of new sources of relevance evidence to search engines leads to an unwanted collateral effect: several new processing costs are added to query processing due to this new information added in the process. Considering the steady growth of the Web, both in content and in number of users, it is clear that not only the quality of the answers, but also computational efficiency, are crucial aspects in search engine design.

We propose and study a new method to deal with this multiple source of relevance evidence, that raises the overall system computational efficiency while keeping a similar final quality of results, called *LePrEF* (Learning to Pre-compute Evidence Fusion). It aims at combining several distinct sources of relevance evidence in a unique, *unified term impact (UTI)* at indexing time. This, as shown in our experiments, leads to a significant decrease in the overhead caused by these new sources, effectively reducing query processing times.

In order to compute those UTIs, we proposed to use a Genetic Programming (GP) approach. As experiments show, GP (when using a special individual selection method also proposed in this work) can achieve results better than the current state-of-the-art Learn to Rank methods in Web search. The use of LePrEF with GP, besides achieving a strong final result quality also as shown in empirical experiments, lead to a significant diminishing in query response times and an increase in throughput. The discussion about LePrEF, including all experiments performed thus far, are presented in chapter 3.

This new proposed fusion technique has a disadvantage: since each submitted query can not be known beforehand, the unified impact is computed for each term independently and at query time possible relationships between the query terms are disregarded. However, there has been an effort in the past few years to find ways to introduce term relationships in the document as a new source of relevance evidence, which led to significant results. Due to this deficiency of LePrEF, we also proposed a new learning to index technique to add co-occurrence information in search systems at indexing time.

This machine learning technique aims at automatically detecting pairs of terms that have sentence function of *phrasal terms* in text documents. Phrasal terms are sequences of words that function as a single concept, such as phrasal nouns, verbs, adjectives and adverbs. Thus, phrasal terms are ordered sequences of terms with specific meanings, which might be totally distinct to the meaning of its individual compounding words. Examples of phrasal terms are “hot dog”,“look after”, “artificial intelligence”, etc.

The task of finding phrasal terms can lead to advantages in various applications related to natural text processing, such as classification, text search and keyword selection (tau Yih et al., 2006). The main objective of identifying phrasal terms and indexing them is to better represent documents indexed by the search systems, taking into consideration the relationships between the words. Terms can be part of compositions, creating new expressions with completely different meanings from those of the individual words, leading to possible losses of information if these compositions are disregarded. A previous knowledge of which sets of words have a specific meaning when placed together in order can be used in various practical text-based application scenarios, possibly enriching the overall quality of such tasks.

The enrichment of traditional IR methods is an good example of the application of phrasal term detection methods. Most traditional IR models represent textual information as a "bag-of-words", i.e., a set of completely independent terms (Baeza-Yates and Ribeiro-Neto, 1999), disregarding the order which the words appear in the documents. While that approach achieved somewhat good results, it is clear that considering the words present in a text document as completely independent of each other is not a faithful representation of their relationships. Sets of words can be part of larger compositions, giving birth to different expressions with their own meanings, and ignoring this can lead to errors due to possible ambiguities.

Thus, in this thesis, we also proposed a method to automatically detect such phrasal terms in text collections, and measure their impact on search with the Vector Space Model(VSM) (Baeza-Yates and Ribeiro-Neto, 1999). To do so, we deployed a supervised machine learning method, Support Vector Machines (Joachims, 2002) in order to automatically detect the meaningful phrasal terms. Experiments presented in this work show that the precision achieved by our method is up to 94%, and up to 36% of improvement in search quality, when measured by MAP. These results and the overall discussion about our method and its results are presented in

Chapter 4.

Finally, we study the impact of applying the two applications of machine learning together in search systems, thus creating an index that contains phrasal terms and adopts the evidence fusion strategy proposed by us. The results of experiments combining the two techniques indicate that we can reach high quality results while keeping the computational costs fair smaller than the ones obtained by previous efforts of applying machine learning to search systems at query processing times. These results also indicate that the new alternative of applying machine learning methods at indexing times is a promising approach, opening opportunities for future research in this area.

1.1 Contributions

Most of the research done using Machine Learning to enhance search tasks is focused on enhancing Query Processing directly, for instance (Pôssas et al., 2005) by applying Learning to Rank at query processing times or learning meaningful bigrams and using them to modify queries and the ranking also at query processing times. In this thesis we propose two new techniques aiming at enhancing search engines that are deployed and affect the indexing process, which is mostly unheard of in the literature.

One of the proposed Machine Learning methods is LePrEF, a method to create Unified Term Impacts (UTIs) during indexing time and use these UTIs to compute the ranking on search engines. This is done by applying a Learning to Rank algorithm (Genetic Programming) to learn the best function to fuse all different sources of relevance evidence available at indexing time in a fashion that, when used in query processing, yields a top quality ranking.

While the concept of indexing the impact of a term instead of its raw frequency is not new, the notion of using a single importance value to represents a large body of sources of relevance evidence was not previously proposed in the literature, being in itself a new concept proposed in this thesis.

We also take this approach one step further in our method, not only condensing several sources of evidence into a single unified term impact, which leads to smaller indexes, but in practice performing most of the Learning to Rank method during the indexing of the text collection, instead of during query processing, leading to further performance gains. Further, the achieved quality results of this evidence fusion was also competitive when applied to the LETOR dataset, and even outperformed traditional Learning to Rank approaches when applied to a training set with a larger number of evaluated documents.

We also proposed a modification in the traditional approach to use Genetic Programming for Learning to Rank: we modified the best candidate selection stage present in GP by performing many runs on the same training data with different random seeds, instead of single runs as usual. We then chose the candidate with the best results in the training and validation sets as our best individual. This led to visible gains in quality of results when using learning to rank with GP, since the variation of results observed due to different random seeds was not negligible. This GP variation outperformed all other learning to rank methods considered as state-of-the-art. We submitted these results to the LETOR dataset managers Liu et al. (2007), and they stated that these results will be made available to other researchers as one of the baseline methods available in LETOR to compare learning to rank techniques.

The LePrEF method and its experimental results were published in the Journal of the American Society for Information Science and Technology (JASIST), in the paper entitled: ”LePrEF: Learn to Pre-Compute Evidence Fusion for Efficient Query Evaluation” (da Costa Carvalho et al., 2012).

Besides LePrEF, we also proposed a machine learning method to detect meaningful phrasal terms in the text collection. The proposed method uses co-occurrence information to infer which of the bigrams present in a text collection are phrasal terms, i. e., have actual meaning when in a composition.

One of the main advantages of our detection method is that it is grammar independent, since it uses only co-occurrence information. Our experiments showed that it achieved precision in the detection in varied domains of documents up to 94%, such as Web pages and newspaper articles, and it is not unfathomable that such performance could repeat itself when applying it to datasets in different languages.

Not only we presented experiments showing that the precision and recall of this detection is remarkable, but we also presented experiments showing how the inclusion of phrasal term information in search indexes led to better search results. Gains in MAP were up to 23% when considering only the queries that have phrasal terms. The method and its experimental evaluation results were published by the Journal of Information and Data Management (JIDM), in the paper entitled ”Using Statistical Features to Find Phrasal Terms in Text Collections”.

We also performed experiments showing the overall impact of our two machine learning methods when deployed together. We believe that such experiments are important to show the overall impact of both techniques proposed in this doctorate may have when deployed together in a single search system, to give a better understanding of the extent that the contributions of this work might have in real systems.

The remainder of this thesis is structured as follows: In chapter 2, we present basic concepts and related works to the subject of this thesis. In chapter 3, we present LePrEF, our proposed method to perform evidence fusion during indexing time. In chapter 4, we present a method to automatically detect meaningful bigrams in text corporas, and evaluated its impact on text search. In chapter 5, we further expand our investigation to evaluate the impact of phrasal terms in multiple evidence scenarios, and its possible improvements in the results obtained by LePrEF.

In chapter 6, we present our conclusions and future research directions.

Chapter 2

Basic Concepts and Related Work

In this Chapter, we present some basic concepts and previous work that are related to research presented in this thesis, in order to facilitate the understanding of readers that are less well versed in the scientific literature pertaining the subjects presented. For a clearer presentation, we divide this Chapter into the following topics: In Section 2.1, we present an overview of machine learning methods. In Section 2.2, we present the literature revision related to pre-computed impacts for evidence fusion in search engine indexes. In Section 2.3 we present the most recent and important previous work pertaining the detection and application of the use of meaningful phrasal terms in information retrieval tasks.

2.1 Machine Learning

Machine learning is the name given to a collection of computational techniques that intend to infer information based on patterns found in data. Basically, a machine learning (ML) algorithm receives data as input and can yield patterns and predictions based on the features of underlying mechanisms that generated this data, being, in a more rough sense, an algorithm that **learns**.

In a broader view, ML algorithms aim at creating a generic model from samples, i.e., to apply the patterns that were learned in data in new examples, not different from how a human being learns from experience.

In regard to its learning phase, machine learning techniques can be divided into three types (Baeza-Yates and Ribeiro-Neto, 1999): (1) Supervised learning, where the learning function requires training data as input, which is usually labelled by human specialists, (2) Unsupervised learning, which are a set of algorithms that do not need human labelled data, but are not plausible in all learning situation, and (3) Semi-supervised learning, which combines a small amount of labeled data with a larger amount of unlabeled data to improve predictions. In this Thesis, we deployed supervised techniques, which we believe are more suitable to the problems at hand (Further discussion is present in Chapters 3 and 4).

In this thesis we are interested in two specific applications of machine learning: classification tasks and learning to rank tasks (Baeza-Yates and Ribeiro-Neto, 1999, Witten and Frank, 2000). In classification tasks, the main objective is to, given a set of user defined classes, to use a number of features of the objects being classified to distinguish which class each probably belongs to. Roughly speaking, the most important tasks when designing supervised machine learning techniques is obtaining the manually labeled training data and proposing which features are more likely to distinguish which class does an element belong to.

In learning to rank tasks, the main objective is not to classify the elements into previously defined classes, but to provide an ordering for them. There are two main approaches to this problem: (1) using pairwise comparisons between the ranked elements in order to assess the ranking quality, as is done by methods such as RankSVM and RankBoost and (2) generating scores for each element and then ordering them by it, as is done with Genetic Programming.

In this thesis we deployed two machine learning approaches, according to the problem that we were interested to solve: Support Vector Machines for the problem of classifying bigrams as phrasal terms or not, and Genetic Programming to create the unified term impact (UTI), which is then used to rank search results. While it might seem questionable why we used those methods instead of others present in the literature, the experimental results obtained and our own preliminary experiments showed that, among the best Machine Learning techniques, it made little difference if one was chosen instead of another, achieving similar final results.

2.2 Pre-Computed impacts

In our work we use genetic programming (GP) as a tool for learning unified pre-computed impacts from several distinct sources of relevance evidence. Several previous work in literature (de Almeida et al., 2007, Fan et al., 2004a,b, Silva et al., 2009, Trotman, 2005, yuan Yeh et al., 2007) apply GP to discover ranking functions. For instance, success has been reported in applying GP to find ranking functions optimized to specific queries in the information routing task (Fan et al., 2004c).

In this work we also use GP as the learning method, but instead of applying the learning to rank method at query processing time, we apply it at indexing time to combine indexes related to several sources of evidence into a single index. To achieve such a goal, we use GP to learn weights for each occurrence of a term in a document, as is done by Fan et al (Fan et al., 2004b), but using the learning process to compute a unified impact and store it in inverted lists at indexing time, instead of using it when processing queries. We investigate the impact of our approach both in effectiveness and efficiency of a search engine, presenting also a discussion about future research related to this topic.

We have chosen to adopt GP instead of other machine learning methods to pre-compute term impacts because: (i) GP has proven to be a quite competitive machine learning approach for ranking results in search systems for textual databases (de Almeida et al., 2007, Silva et al., 2009). (ii) It produces as a result functions that compute a numeric value. Such functions can thus be directly applied to compute the term entry impacts. Further, we also show how to easily adapt such functions produced by GP to scale the pre-computed scores to integer values. Notice however that other machine learning approaches, such as the method applied in Adarank (Xu and Li, 2007), could be adopted as the basis for learn to pre-compute impact methods.

Besides GP, other learning techniques have been applied successfully to information retrieval problems. Several previous research articles aiming at learning how to rank in search engines have been deployed in the last decade (de Almeida et al., 2007, Fan et al., 2004a, Freund et al., 2003, Joachims, 2002, Silva et al., 2009, Trotman, 2005, yuan Yeh et al., 2007). The common feature in these works is that they try to learn how to generate an optimum ranking for any possible new query, given a set of features and training data composed by labels representing the importance of a document with respect to a query.

The work presented by Joachims (Joachims, 2002) shows how to learn good ranking functions by using click-through information from the results that have been returned to the user. This paper presents a SVM algorithm to perform the ranking task instead of the classical classification problem, where it is applied. Boost strategies, like RankBoost (Freund et al., 2003) have presented good results and shown that the combination of multiple weak ranks can lead to a highly accurate ranking. In all mentioned cases, the learning function is applied at query processing time.

A benchmark dataset to facilitate a fair comparison between different learn to rank algorithms was proposed in (Liu et al., 2007). This dataset is referred as LETOR. This benchmark also describes a set of features to be used for learning to rank and the quality results obtained by some algorithms. The list of the results obtained by the best methods over the same set of collections and features, expressed through different performance measures, is made available, making it easier to compare a new approach with the state-of-the-art rank approaches. Experiments were made with two of the most well-known learn to rank methods at the time, RankBoost (Freund et al., 2003) and RankSVM (Joachims, 2002). Since them, many new versions of the database and baselines were made available, and more recent versions added experiments with Listnet (Xia et al., 2008) and Adarank (Xu and Li, 2007). All those methods results are publicly available and are normally used as baselines when using LETOR.

Previous learn to rank methods almost always focus solely on the effectiveness (quality) issues, ignoring the efficiency (computational cost) implications of their uses in actual search engines (Wang et al., 2010). Thus, Wang et al. (Wang et al., 2010) proposed a modification in WSD (Bendersky et al., 2010), a previously proposed learn to rank method, to add a feature pruning threshold. This is achieved through the MEET metric, which is a trade-off between computational cost and quality. LePrEF, on the other hand, proposes a completely distinct optimization strategy, since it **pre-processes** all features at indexing time, obtaining a single feature index. The results on (Wang et al., 2010) reinforce our intuition that a large number of processed features (sources of relevance evidence) has a significant efficiency impact in search engines.

In a recent work, Cambazoglu et al. (Cambazoglu et al., 2010) proposed optimization strategies based on additive ensembles that are specific to process queries when using learning to rank methods. The authors present a case study using RankBoost. Their proposal is to perform short-circuiting score computations in additive learning systems. The strategies are evaluated over a state-of-the-art machine learning system and a large, real-life query log, obtained from Yahoo!. The method we propose here can take advantage of their proposal to produce fast early termination ranking strategies.

The use of simpler, pre-computed impacts of terms in documents in order to reduce computational costs was proposed by Anh et al. (Anh and Moffat, 2002), where instead of storing term frequency and other data necessary to the vector-space model (VSM) computation, the authors proposed to store the pre-computed VSM impact (quantized and scaled to integer values between 1 and 32) in order to reduce the number of arithmetic computations during query processing. Later, Anh et al. proposed a new pre-computed impact (Anh and Moffat, 2005), defining this impact qualitatively rather than quantitatively, in a document-centric approach, achieving better retrieval effectiveness levels and more pronunciated computational gains, due to the use of a smaller (between 1 and 8) scale for their impacts. In a third research effort, Anh et al (Anh et al., 2008) also investigated the possibility of pre-computing scores when using BM25 as the ranking method.

Our proposed method of learning to index, LePrEF, can be considered as taking this approach to the next level. It uses machine learning to pre-compute impacts, providing a framework to pre-compute impact with any set of evidences available at indexing time. Further, it calculates an impact not only for the document text and its fields, but for all possible sources of relevance evidence available at indexing time.

2.3 Phrasal Terms

Zhang et al. (Zhang et al., 2007) tackled the problem of noun phrase detection and its possible application to enhancing search engine queries. Noun phrases can be considered a subset of phrasal terms, since phrasal terms do not necessarily have to be nouns. The method proposed by

Zhang uses a number of different datasets, such as Wikipedia[1] and WordNet (Miller et al., 1990), natural language parsers and Google search[2] as sources of evidence, combined in order to detect noun phrases. This work provides good evidence that a subset of phrasal terms (noun phrases), can lead to improvements in text search. However, while the objective of this work is similar to ours (besides the broader spectrum of phrasal terms in comparison to noun phrases), the use of external databases and natural language processing techniques make the method unsuitable for many information retrieval scenarios, due to its high computational cost and dependency from external sources. In contrast, the method we propose in this chapter relies only on the text collection itself to find phrasal terms, and this processing can be done offline at indexing time, i.e., prior to the actual use of the text collection.

Mladenik et al. (Mladenic and Grobelnik, 1998) made experiments using n-grams (up to 5-grams) to improve the performance of a Naive Bayes classifier. The authors found out that the highest improvement was achieved when adding 2-grams to the classification, and also that n-grams with n larger than 3 actually had no positive influence in the classification.

Tesar et al. (Tesar et al., 2006) proposed the use of 2-itemsets (selected sets of two words co-occurring in documents) instead of *bigrams* (sequences of two words) to enhance classification, and made experiments comparing the use of both with a number of feature selection techniques. While the results achieved for both bigrams and 2-itemsets were superior to the ones achieved when considering only unigrams, the results obtained by adding bigrams were consistently superior to those with 2-itemsets, indicating that bigrams are a better option for classification purposes since computing 2-itemsets is also more expensive.

Based on the assumption that the use of all the bigrams present in the pages would add noise

[1]www.wikipedia.com
[2]www.google.com

to the categorization, Tan et al. (Tan et al., 2002) applied a feature selection method to select only bigrams that are likely to be useful for the classification. The authors proposed an algorithm based on the information gain metric, combined with some frequency thresholds to select these bigrams, achieving significant gains in classification. This approach, however, is focused on classification and can not be trivially adapted to be used in other information retrieval tasks, in contrast with our method. Also, it is orthogonal to our method, since it is not interested in finding meaningful bigrams , but simply bigrams that are good class discriminators.

Finally, it is interesting to mention that in (Ekkerman and Allan, 2003), Ekkerman et al. claim that, up to that point, many efforts have been made to improve text categorization by the use of bigrams, with little or no observable improvement in the classification results. He also proposed a method to include bigrams information in the classification, based on distributional clusters, but this method yielded statistically insignificant improvements in the quality of results. As discussed above, subsequent works have had more success in this task.

Wang et al. (Wang et al., 2012) revisited the problem of detecting n-grams to enhance search tasks with a different approach: Instead of first detecting the n-grams that have sense together, as we did, and then evaluate their uses in search tasks, they have skipped this detection on simply finding the n-grams that most probably would lead to gains in search, irregardless of its inherent meaning. However, while our proposed phrasal term detection method was already published over a year before the publication of the paper, even though they cite our method, no comparison was done with it.

Chapter 3

Learn to Pre-Compute Evidence Fusion

In the last two decades, numerous research articles have been published proposing new sources of relevance evidence to enhance search in large textual databases, such as those used by Web search engines. Examples of these sources are the link structure of the web, the anchor text found in web links, the text extracted from URLs, click-through data and personalized user information, among several others. The use of these more sophisticated sources of relevance evidence, in contrast with just using the text content of the web pages, yielded expressive quality gains in search engine results.

A closely related research problem that has also been under intensive investigation recently is how to best use these several sources of evidence together to enhance the search results. First steps in this direction were to simply combine them linearly or using probabilistic models (Calado et al., 2003). While these attempts yielded good results, soon more sophisticated approaches arose. One of the most successful of those approaches is to adopt machine learning techniques to automatically construct a suitable ranking model by properly combining several distinct sources of evidence. Examples of methods that follow this *learn to rank* approach are SVM Rank (Joachims, 2002), RankBoost (Freund et al., 2003) and the methods based on Genetic

Programming (de Almeida et al., 2007, Silva et al., 2009).

A problem that derived from combining a large number of sources of relevance evidence to produce a single ranking is the increase in computational costs at query processing time. This problem gets even worse with the continuous growth in the size of the search engine document database and its indexes, as well as the growth of the number of its users and queries posed to it. This problem has a direct impact in the response time perceived by users, which is always a matter of high priority.

In this work, we propose a new method for combining several distinct sources of evidence which produces high quality ranking results while being efficient computation-wise. Our method, called *LePrEF* (Learn to Pre-computed Evidence Fusion), computes a *unified pre-computed impact* value for each term found in each document prior to query processing, at *indexing time*. These values are indexed and later used for computing document ranking at query processing time. By doing so, our method effectively reduces the query processing to simple additions of such unified impact values.

The idea of adopting pre-computed impacts when indexing large textual databases has been studied before in previous work (Anh and Moffat, 2006, Anh et al., 2008). However, as far as we know, in this work we present the first attempt to combine learning to rank methods with the idea of using pre-computed scores. As shown through experiments, this combination results in a flexible pre-computed score scheme. Although we only evaluated it in this thesis with an specific set of sources of relevance evidence available in a benchmark Web page collection, it can easily be applied to other sets of relevance evidence available at indexing time, and to other textual database collections. One of the drawbacks of our method is that features not available at indexing time should be combined separately. However, its achieved quality results are comparable to what is achieved by other learning to rank approaches, with the advantage of the efficiency gains that arise from the use of pre-computed impacts.

To compute *unified pre-computed impacts*, we deployed a Genetic Programming technique, which has been shown to achieve very satisfactory results in traditional learn to rank tasks. Further, using it to pre-compute impacts leads to, besides a strong rank quality, visible efficiency advantages at query processing time even compared with simple evidence combination methods, since it fuses all sources of evidence at indexing time.

We also present a simple alternative to transform the unified pre-computed impacts to integer values, thus avoiding floating point operations at query processing time and the overhead in compressing floating point numbers, in comparison to integers. We show through experiments that when converting pre-computed impacts to integer values we can further reduce the size of the pre-computed index by using simple and low cost compression techniques previously proposed in literature. In order to diminish the possible negative impacts of such transformation, we incorporate it in the training stage of our genetic programming framework, obtaining final pre-computed impact integer values with a low loss in quality of ranking.

Our motivation to develop *LePrEF* comes from the observation that combining several sources of evidence at query processing time is always an expensive task, regardless if the combination is performed by using a simple combination method or state-of-the-art methods, such as learn to rank. For instance, the learn to rank benchmark collection LETOR, as proposed in (Liu et al., 2007), uses 46 features. Each feature must be fetched and in some case computed individually to only then apply a learn to rank method. Further, each specific learn to rank method may also incur in new computational costs. LePrEF avoids these costs by pre-computing the fusion of evidences at indexing time.

3.1 Pre-Computed Evidence Fusion

The LePrEF method adopts the concept of *Unified Term Impact (UTI)*. A UTI is a single value that represents the importance of a query term to a given document in the search engine database. This value is produced through a fusion of all sources of relevance evidence regarding that term-document pair at indexing time. More precisely, the set of indexes for all sources of evidence are fused into a single inverted index containing the UTI entries.

In Figure 3.1 we contrast the traditional indexing approach (a), in which the outcome consists of several indexes, one for each source of evidence, with the approach adopted by our method (b), in which a single inverted index containing the UTI entries is produced as the result of fusing all sources of evidence. Notice that, for didactic purposes, we illustrate the traditional indexing as producing a separate index for each source of evidence in Figure3.1(a). In practice, all the evidence information may be stored in a single global index. In any case, still the traditional indexing process stores a separate piece of information about each individual source of evidence in this index. Further, storing all information in a global index requires an overhead to indicate which features are present on each index entry. In our experiments using compressed indexes, such overhead resulted in losses both in performance and storage space.

The LePrEF method has a considerable positive impact on the processing of queries submitted by users, since it avoids the fusion of several sources of evidence at query processing time. The advantages of using LePrEF are even more appealing if we consider how previously proposed learning to rank methods work: the query processing in Web search engines that use learning to rank is usually performed in a two-phase scoring scheme (Cambazoglu et al., 2010), as illustrated in Figure 3.2(a).

In the first phase, a *candidate selector module* uses a simple scoring technique for selecting a

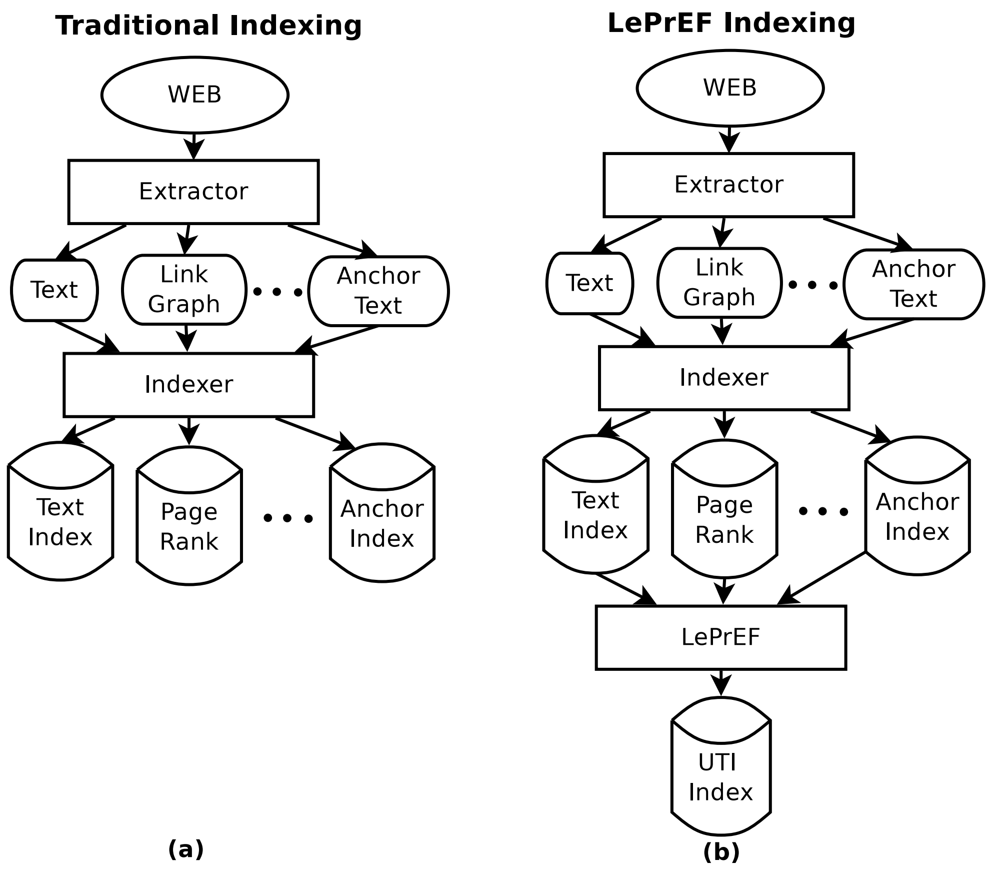

Figure 3.1: (a) Traditional search engine indexing processing and (b) the indexing processing using LePrEF.

small subset of potentially relevant documents from the entire collection. The candidate selector can, for instance, apply the BM25 algorithm to one or more sources of relevance evidence, taking the top answers obtained to compose the candidate documents. The query processor must then fetch information corresponding to all sources of evidence related to the candidate documents in order to re-score these documents by using a more sophisticated learn to rank strategy. The final ranking is determined by the document scores computed in the second phase.

As illustrated in Figure 3.2(b) our method not only can avoid this two-phase query processing, but also can lead to an increase in query throughput by fusing the sources of evidence at indexing time. Indeed, in our case, the final score computation is reduced to a very simple sum of the UTI values available on the index entries for each term found in the query.

Our experiments show that using LePrEF in a single phase results in a quite slightly smaller quality of ranking results when compared to state-of-art machine learning approaches. However, as also shown in the experiments, this slight diminish in quality translates into a significant reduction in the overall computational costs of query processing.

It is important to notice that LePrEF cannot be used to fuse sources of evidence that are only available at query processing time. This is the case, for instance, of personal information about the preferences of each search engine user. These type of sources of evidence must be fused by another method at query processing time, thus by the use of LePrEF in addition to a second phase to process these evidences. However, LePrEF still yields a reduction in query processing costs by fusing all the evidences that can be computed at indexing time, and by reducing the costs for the first phase of query processing.

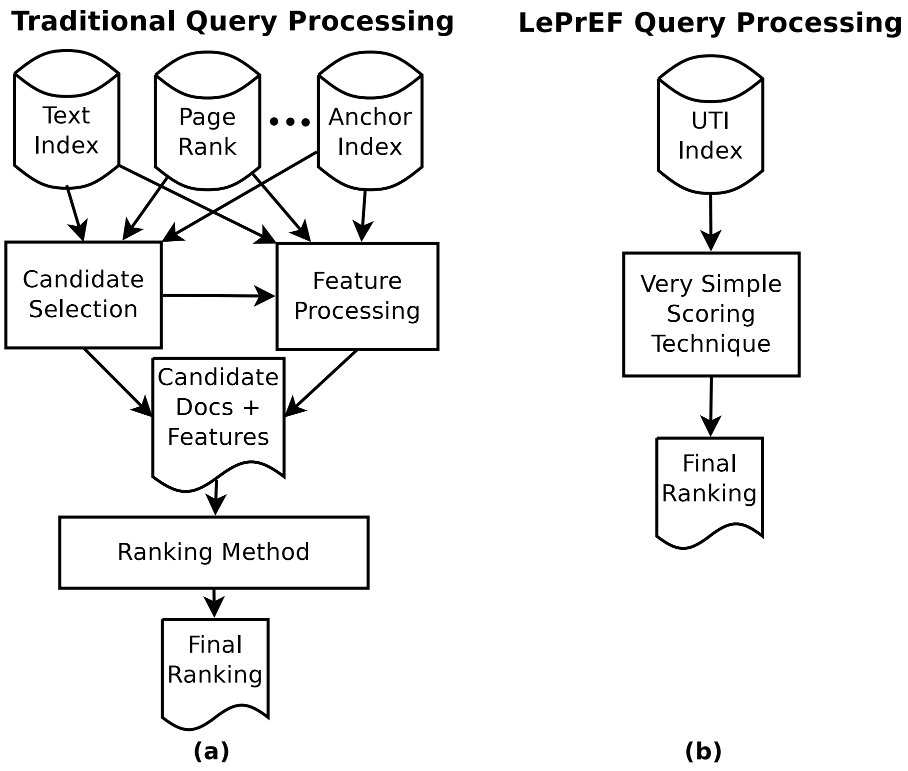

Figure 3.2: (a) Traditional query processing when using learn to rank methods, and (b) query processing when using LePrEF.

When using the traditional methods that fuse evidence at query processing, the computational efforts for the ranking computation at query processing time may grow as more sources of relevance evidence are considered, specially when these added features are also taken into account in the selection of candidate documents. For instance, when adding any new textual source of relevance evidence, the first phase of query processing should take such information into account to select the set of potentially relevant documents. As a further detailed example, suppose a fairly common scenario in which, besides the text of the body of a Web page, the text extracted from URLs, the text of the title and the anchor text information are also used as sources of relevance evidence. In this case, a simple score technique should also be applied to each of these new sources of relevance evidence at the first phase of query processing, adding more costs to query processing. Of course there are many methods to reduce the computational cost of query processing, such as pruning (Anh and Moffat, 2006, de Moura et al., 2008) and cache techniques (Blanco et al., 2010, Saraiva et al., 2001). However, the query processing time will ultimately still be affected by the computation of scores provided by various sources of evidence in both phases.

Conversely, when LePrEF is deployed, the use of pre-computed, unified term impacts (UTIs) leads to a much simpler query processing, since it is no longer necessary to process each source of evidence separately. This advantage results in a reduction in the number of operations performed by the query processor since it is, in practice, dealing with a single and unified representation of the sources of relevance evidence adopted.

The usage of UTI indexes presents two additional advantages that are worth mentioning: (1) Most dynamic pruning techniques are developed to prune entries of a single inverted index of term impacts (Anh and Moffat, 2006, de Moura et al., 2008, Persin et al., 1996), which would allow the direct application of such methods to the index generated by LePrEF. When using multiple indexes, the pruning strategies may be less effective, since, as pointed in previous work (de Moura et al., 2008), the set of documents with greater impact according to each source of evidence are expected to be distinct, while most of the relevant documents are in the intersection of these sets. When using the UTI index, we already have access to the final impact of each term for each document regarding all sources, thus providing more accurate information to improve the performance of pruning methods. We plan to quantify and investigate such advantage in a future work. (2) The UTI index entries, storing a single value instead of multiple values, lead to a smaller search index, which results in smaller seeks and overall smaller reading times, affecting also the cache policies in cases where the index is stored in disks.

On the other hand, the usage of LePrEF has a disadvantage. The method requires a complete new index generation to be carried out whenever a new fusion formula is generated. As the learning to rank processes are adjusted to the collections over which they run, and considering that the web search engine databases continuously change, there is a necessity of adjusting the learned functions from time to time. It is important to consider, however, that this restriction does not hinder the use of the UTI indexing strategy in practice, since the reasons for considering a term occurrence important do not change as fast as the database changes. We consider that this issue also requires a specific study in our future works.

As described above, the fusion of the sources of evidence for generating the UTI index ideally requires the use of a learn to rank method. The main requirement regarding such a method is that it should be capable of generating as output a function f that takes a set of values $e_1, \dots, e_n$, each one corresponding to a distinct source of evidence, and gives a single value u that will be used as the UTI value.

In our work we decided to deploy a Genetic Programming (GP) learn to rank approach. GP was adopted in our study because it is known to have a solid performance as a learn to rank method. Indeed, previous work using Genetic Programming for learn to rank (de Almeida et al., 2007, Fan et al., 2004a, Silva et al., 2009) have shown that it achieves good precision and recall levels in comparison with the state-of-the-art learn to rank methods. We provide more insight about the quality of GP as a learn to rank method by applying it to a benchmark learn to rank collection. In this benchmark, GP produced the best performance when compared to the state-of-art learn to rank methods.

Additionally, the functions generated by GP can be directly applied to compute UTI values. The application of the UTI strategy using other learning to rank methods would not be direct, requiring an adaptation of those learning methods to produce UTI impact values as output. We anyway plan to investigate this possibility in future works.

3.2 Computing UTI Values With GP

The Genetic Programing (GP) process adopted in LePrEF to produce the UTI index follows the same general steps adopted by the GP process used by traditional learning to rank methods. The main differences are in the type of function generated in each process, that is, the individuals evolved, and in the way these individuals are evaluated. For sake of completeness, in this section we explain how the GP process works in both methods, highlighting the differences in each case.

The general GP process is described in Listing 3.1. GP is basically an iterative process with two phases: *training* (Lines 5–13) and *validation* (Lines 14–16). For each phase, a set of queries and documents is selected from a distinct collection, which we call the *training set* and the *validation set* (de Almeida et al., 2007), respectively.

The process starts with the creation of an initial random population of individuals (Line 5) that evolves generation by generation using genetic operations (reproduction, crossover, and

Listing 3.1: General GP process used both in learn to rank and in LePrEF

```
 1 Let $\mathcal{T}$ be a training set of queries;
 2 Let $\mathcal{V}$ be a validation set of queries;
 3 Let $N_g$ be the number of generations;
 4 Let $N_b$ be the number of best individuals;
 5 $\mathcal{P} \leftarrow$ Initial random population of individuals;
 6 $\mathcal{B}_t \leftarrow \varnothing$;
 7 For each generation $g$ of $N_g$ generations do {
 8     $\mathcal{F}_t \leftarrow \varnothing$;
 9     For each individual $i \in \mathcal{P}$ do
10         $\mathcal{F}_t \leftarrow \mathcal{F}_t \cup \{g, i, \mathit{fitness}(i, \mathcal{T})\}$;
11     $\mathcal{B}_t \leftarrow \mathcal{B}_t \cup \mathit{getBestIndividuals}(N_b, \mathcal{F}_t)$;
12     $\mathcal{P} \leftarrow \mathit{applyGeneticOperations}(\mathcal{P}, \mathcal{F}_t, \mathcal{B}_t, g)$;
13 }
14 $\mathcal{B}_v \leftarrow \varnothing$;
15 For each individual $i \in \mathcal{B}_t$ do
16     $\mathcal{B}_v \leftarrow \mathcal{B}_v \cup \{i, \mathit{fitness}(i, \mathcal{V})\}$;
17 BestIndividual $\leftarrow$ $\mathit{applySelectionMethod}(\mathcal{B}_t, \mathcal{B}_v)$;
```

mutation) (Line 12). The process continues until a stopping criterion is met. In the case of Listing 3.1, the criterion is the maximum number of generations of the evolutionary process.

In the training phase a fitness function is applied to evaluate all individuals of each generation (Lines 9–10), so that only the fittest individuals are selected to continue evolving (Line 11).

In the case of learning to rank, each individual represents a weighting function to assign a score to each document given a query. The fitness of an individual corresponds to the quality of the ranking generated by the individual for each training query.

In the case of LePrEF, each individual is an evidence fusion function which is adopted to produce a UTI value for each entry in the inverted index. The training queries are processed using this UTI index to produce a ranking of the set of training documents. To evaluate the fitness of each individual, in LePrEF we sum, for each term t found in the collection, the UTI impacts of this term for each document in the collection. The scores obtained are then used to rank the documents, and this final ranking is used to compute the fitness of each individual.

After the last generation is created, to avoid selecting individuals that work well in the training set but do not generalize for different queries/documents (a problem known as *over-fitting*), the validation phase is applied. In this phase, the fitness function is also used, but this time over the validation set of queries and documents (Lines 15–16). Individuals that perform the best in this phase are selected as the final solutions (Line 17).

Individuals

An individual is represented by terminals and functions, organized in a tree structure, as shown in Figure 3.3. Terminals, or leafs, contain information obtained from the sources of relevance

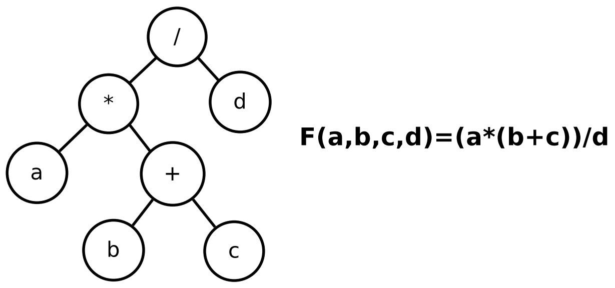

Figure 3.3: An individual from the GP process.

evidence. For this, main ranking formulas published on IR literature can be used. In our experiments we adopted as terminals the values of the features presented in Section 3.4.1. In addition to these, we also use constant values in the range [0..100]. As functions in the inner nodes, we use addition ($+$), multiplication ($*$), division ($/$) and logarithm ($\log$). We use the genetic operators of reproduction, crossover and mutation as proposed by Almeida et al. (de Almeida et al., 2007).

Fitness Function

The fitness function must measure the quality of the ranking generated when using a given individual. In preliminary experiments, we evaluated our method with two different fitness functions: MAP and mean NDCG. When using mean NDCG as the fitness function, LePrEF yielded better overall results both in terms of mean NDCG and MAP, thus we ultimately adopted mean NDCG as the fitness function in the experiments.

Selection of the Best Individuals

The validation step in our experiments was performed as proposed in (de Almeida et al., 2007). According to this approach, the choice of the best individuals is accomplished by considering the average performance of an individual in both the training and validation sets, minus the standard deviation value of such performance. This method is called AVG$_\sigma$. The individual with the highest value of AVG$_\sigma$ will be selected as the best.

More formally, let t_i be the training performance of an individual i, let v_i be the validation performance of this individual, and let σ_i be the corresponding standard deviation value of $(t_i + v_i)$. The best individual is selected by:

$$\underset{i}{argmax}((t_i + v_i) - \sigma_i) \tag{3.1}$$

Notice that a smaller value of σ_i has a larger contribution to the selection, thus giving preference to individuals that have a more regular performance in the queries adopted in the training and validation sets, while $(t_i + v_i)$ also gives preference to the ones that achieve high performance in both sets.

Due to the inherently randomness present in the GP process, and in order minimize the chances of finding an underperforming best individual, we take the GP one step further: Instead of a single process, we run N processes with distinct random seeds, and pick the the best individual (according to AVG$_\sigma$) among the ones generated by these N runs. By doing so, we diminish the chances of a single seed leading to a below average performance. As shown in Section 3.4, this strategy resulted in selecting individuals that perform above the average of all runs.

3.3 Integer UTIs

In our initial formulation of LePrEF, the UTI entries are by definition real numbers. While this implicates that the range and the precision of UTI values are very large, the usage of real numbers can lead to two disadvantages: first, a sum of real numbers is computationally more expensive than a sum of integers, requiring more processor cycles at query processing time. Second, the usage of real numbers may result in worse compression rates to the UTI indexes in comparison to the usage of integer numbers, which may be an important property, since compression methods are usually applied in search systems (Baeza-Yates and Ribeiro-Neto, 1999).

Due to these two factors, and in order to be able to capitalize on most previously proposed performance enhancing methods, we also propose a modification in the UTI index computation, in order to represent its entries as integer numbers, instead of real numbers.

To implement the discretization procedure we considered two alternative strategies: (1) Perform the training with GP as usual, in the same way done with UTI as a real number, and then perform the discretization process over the final result. (2) Train the GP using as the fitness function the results obtained after transforming the values to integer. To do so, we modified the fitness function in the GP framework to truncate all the UTI index entries generated ($UTI_{integer}(t,d)) = \lceil UTI(t,d) \rceil$). With this change, the fitness function selects the best individuals that generate integer UTI indexes. The intuition behind this later option is that instead of trying to find the best UTI value and then transform it to integer, we train to find the best integer UTI values as part of the learning process. In our experiments, we saw that the results obtained by (1) were extremely erratic, with a very large variation in quality depending on the learned function. Conversely, (2) had a much more solid and overall superior results, due to the fact that the GP process took this discretization into consideration when selecting the best functions.

3.4 Effectiveness Experiments

In this section, we present experiments done to show how LePrEF behaves in comparison to other previously proposed methods regarding the quality of query results.

3.4.1 Experimental Setup

We chose to use LETOR (Liu et al., 2007), a benchmark dataset designed to evaluate ranking methods, to evaluate the quality of ranking results produced by LePrEF. Specifically, we chose to use the MQ2007 subset of LETOR 4.0, since it contains a large number of queries as examples (1692).

MQ2007 subset contains 46 features for documents sampled from the top 1000 retrieved documents by using BM25 on the GOV2 corpus for each query. Some of its textual features can only be obtained at query processing time, being not available to LePrEF. These features are the similarity scores assigned by ranking function BM25, and by three variations of Language Models based functions. Each function was applied to five areas of the documents: the body of the text, the anchor text, the title, the URL, and the whole document. Thus, obtaining 20 features based on ranking score functions that we cannot adopt in LePrEF.

The remaining 26 features of MQ2007 are all available at indexing time. They include TF, IDF, $TF \times IDF$ and length in number of words, each of them applied to the same five parts of the documents mentioned above, thus obtaining 20 features. Besides these features, we also adopted the features PageRank, InLink Count, OutLink Count, Number of Slashes on URL, Length of URL and Number of Children. Further detailed information about these features can be found in LETOR documentation (Liu et al., 2007). We used the metadata present in LETOR database to generate the values for each feature when indexing the GOV2 collection with LePrEF. In LETOR, each feature TF represents a sum of TF values obtained with each query term. In LePrEF we did not perform such sum, using the individual values of frequency of each query term in each document as features. The same happens for IDF and for TF $\times$ IDF.

It is important to notice that most of the information necessary to compute the features that are not adopted in LePrEF, such as BM25, is also available in the 26 features adopted, including information about TF and IDF of terms, and their document length. Thus, the final expected impact in the ranking of the lack of these features is expected to be low. In fact, our experimental results show that the results achieved by LePrEF are quite close in effectiveness to the results achieved when using GP and other learning methods with all the 46 features.

As baseline methods, we use four of the best methods with available results for LETOR4: Adarank (Xu and Li, 2007) (using only NDCG as metric, since their results were superior to MAP in every metric), Listnet (Xia et al., 2008), RankSVM (Joachims, 2002), and RankBoost (Freund et al., 2003). Besides these methods, we use GP as proposed in (de Almeida et al., 2007), with the improvement of taking the best individual from several runs, as done with LePrEF. This baseline was included in order to compare LePrEF to a learning to rank method that also uses GP. In this case we used our own implementation of the method, since there is no public data regarding its performance in the LETOR4 Dataset.

The LETOR4 collection, is by default, split into 5 Folds, allowing for a 5 Fold cross validation scheme for results evaluation, using 3 folds as training queries, 1 as validation queries, and one for testing. Thus in this work (and in all baselines shown), unless stated otherwise, the results presented are the average of the results obtained at each testing fold.

We adopted as evaluation metrics NDCG@N, P@N, mean NDCG and MAP, the same metrics used by LETOR (Liu et al., 2007), in order to readily compare LePrEF to several previously proposed learn to rank methods. For details about how to compute such metrics, the reader can address the LETOR documentation (Liu et al., 2007).

Genetic Programming Setting

The experiments presented were conducted with the lilGP genetic programming distribution (Zongker and Punch, 2006). The methodology and parameter settings adopted follow the ones proposed in a previous article that studied the use of GP for learn to rank (de Almeida et al., 2007).

Table 3.1 presents the values used for all parameters adopted for purposes of comparison with future work.

Table 3.1: Genetic Programming Parameters

Parameter	Value
# of generations (G)	40
Population size (PSize)	1000
Tree depth	17
Tournament size	6
Crossover rate (Rc)	0.85
Mutation rate (Rm)	0.05
Reproduction Rate (Rr)	0.10

Since Genetic Programming uses random seeds to create its initial population, its results can be affected by the initialization of the first population. Thus, we performed 10 runs with distinct random seeds. We performed a five fold cross-validation with each run (as split in the LETOR Collection), and chose, for each fold, the individual (UTI function) the best individual among all runs according to the AVG$_\sigma$ selection method. It is important to stress that this same approach can also be applied in a real scenario. While it could be argued that training with different seeds can be expensive, these costs occur **prior** to indexing time and are not prohibitive, leading (as shown in Section 3.4.2) to visible gains in the quality of search results.

3.4.2 Results

Since in our proposed GP framework we decided to run the process several times with different random seeds, selecting the one that achieved the best results regarding our training and validation sets as the best individual, it is important to assess whether this strategy can indeed lead to better results or not. Figure 3.4 shows, for each fold, the average mean NDCG obtained in the test for each individual chosen by our approach and also the average value obtained by running the GP process 10 times. As can be seen, taking the best training and validation individual among 10 runs consistently produced results above the average, which indicates our proposed strategy of running GP with several distinct seeds may result in slight, but consistent, improvements in the final quality of a learn to rank with GP.

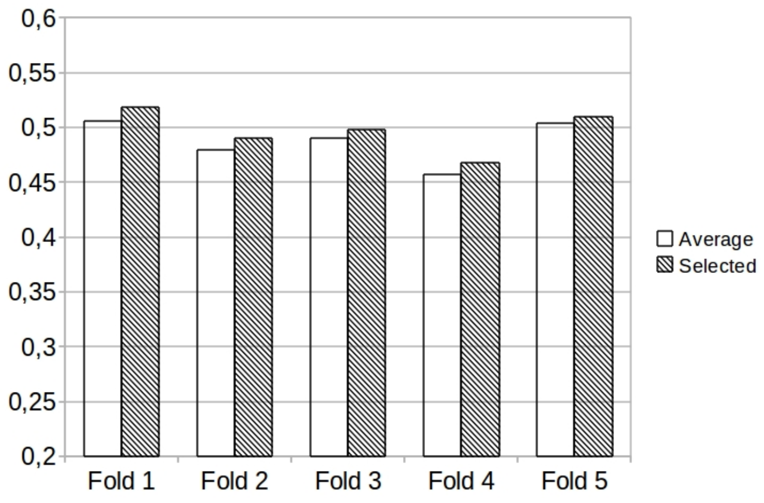

Figure 3.4: Mean NDCG values for the average of the 10 runs and for the run chosen by our validation metric.

Regarding the comparison with the baselines, Figure 3.5 shows NDCG5 and NDCG10 scores achieved by all the methods in the experiments with the LETOR queries. As can be seen, the best overall method was using Genetic Programming (GP), 0.426 in NDCG@5, and 0.443 in

NDCG@10. The second best result was obtained by LePrEF with real UTIs, 0.419 in NDCG@5 and 0.443 in NDCG@10. As can be seen, while there is a visible (and statistically significant) loss in both NDCG@5 and NDCG@10 results when comparing LePrEF against GP, the results obtained by LePrEF in this configuration still stand at top when compared to other state-of-the-art methods proposed in the literature. RankBoost stand above the rest, with NDCG@5 0.418 and NDCG@10 0.446, results that were practically equal in comparison with LePrEF.

When using LePrEF with integer UTIs, as proposed in Section 3.3, the final results were below the ones obtained by real UTIs, as expected due to the loss of information when converting UTIs values from real to integer numbers. However, this diminishing still lead to competitive results when compared with many other proposed learn to rank methods: For instance, while integer UTIs led to NDCG@5 0.409 and NDCG@10 0.436, RankSVM obtained NDCG@5 0.414 and NDCG@10 0.440 and ADARANK NDCG@5 0.410 and NDCG@10 0.437, results that were very near LePrEF with integer UTIs, even though these methods used a richer source of features. It is important to stress that the use of LePrEF with integer UTIS, as shown in Section 3.5, lead to expressive gains in efficiency and competitive results.

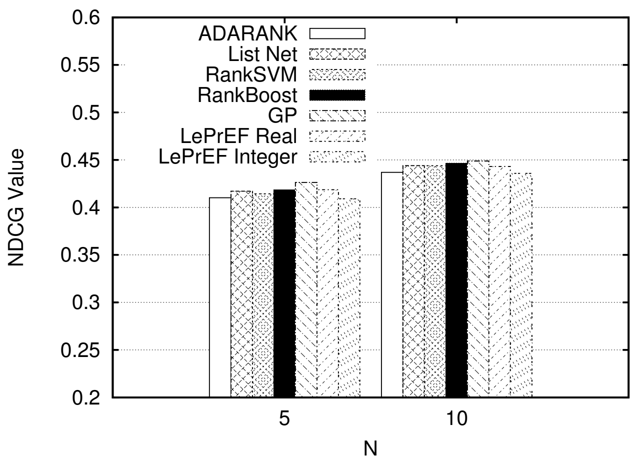

Figure 3.5: NDCG@N results obtained by LePrEF storing real number UTI values in the index (LePrEF Real), LePrEF using integers (LePrEF Integer), the GP learn to rank method, and the four best learn to rank method presented in LETOR benchmark.

Figure 3.6 shows the P@5 and P@10 results obtained by all compared methods. Our GP implementation yielded the best P@5 results, while RankBoost yielded the best P@10 results. As can be seen, LePrEF with real UTIs was again very competitive, yielding P@5 values of 0.417 against 0.422 from GP and 0.418 of RankBoost and 0.413 of SVMRank, In terms of P@10, similar results were obtained, with the difference being that the best performing method was RankBoost (0.386), followed by RankSVM (0.383), GP (0.381), LePrEF with real numbers (0.38) and ADARANK (0.3756). When using our approach to use integer UTIs, we obtained P@5 of 0.411 and P@10 of 0.378, which shows that, while using this approach lead to losses in performance when compared to the **best** methods in the literature, it still lead to competitive performances overall when compared to most proposed methods. It is very important to stress that these very competitive results were obtained when the methods (GP, LePrEF with Real UTIs and with integer UTIs) were trained with NDCG as objective function.

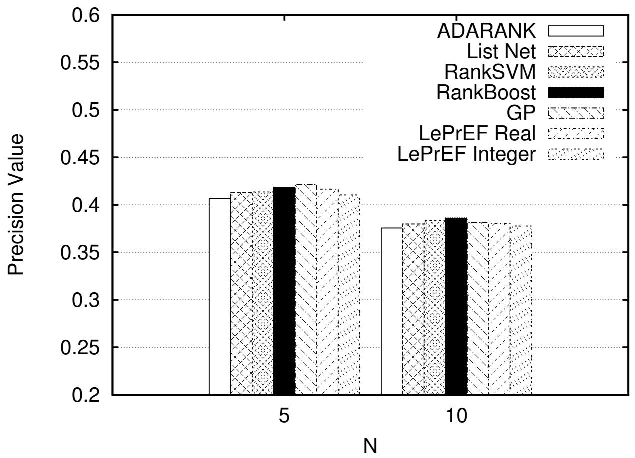

Figure 3.6: P@5 and P@10 results obtained by LePrEF storing real number UTI values in the index (LePrEF Real), LePrEF using integers (LePrEF Integer), the GP learn to rank method, and the four best learn to rank method presented in LETOR benchmark.

Figure 3.7 shows the Mean NDCG and MAP values obtained by the methods, as computed by the LETOR benchmark. As can be seen, the results followed the trends shown by P@N and NDCG@N. The overall best method was GP with our selection method, obtaining Mean NDCG of 0.504 and MAP of 0.469, followed by RankBoost, with Mean NDCG 0.500 and MAP 0.466. LePrEF with real UTIs obtained NDCG 0.497 and MAP 0.461, being indeed very competitive in light of its lack of features in comparison. When using integer UTIs, the results were, as expected from previous results, slightly smaller: Mean NDCG of 0.491 and MAP of 0.455, results on par with those obtained by ADARANK (0.491 of Mean NDCG and 0.46 of MAP) and slightly below RankSVM (Mean NDCG of 0.497 and MAP of 0.464). However, as shown in the next Section, this quality performance is more than compensated by the expressive gains in performance LePrEF in this scenario obtains.

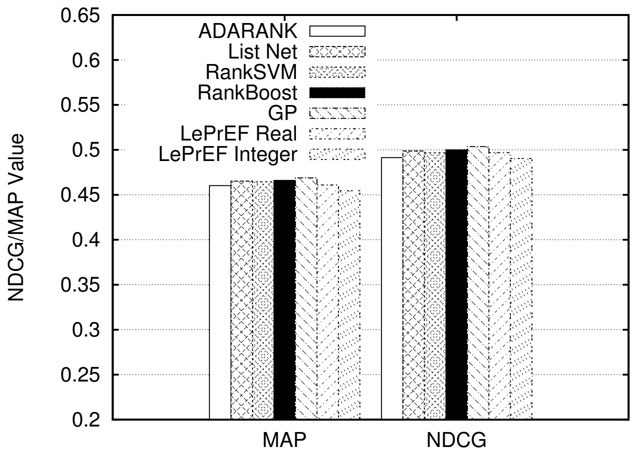

Figure 3.7: Mean NDCG and MAP results using real numbers for UTIs.

3.4.3 Integer UTI

When designing real world applications, there are other important aspects to take into consideration besides the final quality of results. For instance, when designing search engines, the size of the final index is of the utmost importance, with a potentially large impact on the final performance of such systems. In case of LePrEF, when using integer UTIs, the final distribution and range of values can impact in compression rates, as when using variable size encoding schemes such as Elias-δ (Elias, 1975, Zobel and Moffat, 2006), a simple and effective compression method adopted to compress inverted lists.

In order to deliver an insight into this matter, in Table 3.2, we present, for each UTI formula chosen for each fold, the number of bits per entry in the inverted list of the dataset compromised only of the pages present in the LETOR dataset, disregarding the bits used to represent the document id. We use this dataset in order to decide which function we would deploy in Section 3.5, for the efficiency evaluation. While being a summarized dataset, it represents a good sample of the GOV2 collection, thus the relative behavior of the UTI formulas in regard to compression ratios would probably be kept when applying them to larger datasets.

Table 3.2: Average number of bits per entry when indexing the LETOR dataset with the best individual formula of each fold

	# of bits
Fold 1	23,74
Fold 2	51,30
Fold 3	22,83
Fold 4	42,52
Fold 5	13,55

As can be seen in Table 3.2, different formulas can lead to very different compression ratios. While the best performing, the chosen formula from fold 5, used only 13,5 bits per entry, the worst, from fold 2, used 51,30. This difference is mostly due to the fact that the formula from fold 5 can lead to much larger values, which indicates that it is unsuitable for variable-size encoding schemes. It is an interesting fact that there is no direct visible correlation between the number of bits with the final quality results for each fold, being most probably an artifact of the randomness of the GP process.

It is important to notice that, while the average number of bits is much higher than the usual averages shown in the literature, this behavior is expected. The LETOR database is a highly selected database, containing only the top results for each query, basically pruning most less relevant values, which would have a higher chance of achieving smaller UTI values and, thus, using less bits. However, the results displayed in Table 3.2 are important nevertheless to show that different, high quality formulas can lead to very distinct compression ratios.

A in-depth study of the selection of the best individuals regarding both the quality of its final results and its impact in the compression ratio can lead to further improvements in both quality and performance, and we intend to pursue this aspect in future work.

3.5 Efficiency Experiments

We also performed experiments to demonstrate the impact of LePrEF in the computational costs of query processing, comparing our method to baseline methods based on two-phase query processing strategy, based on possible different implementations of multi-feature indexes.

3.5.1 Experimental Setup

In our efficiency experiments we are interested in two aspects: (1) Impact of using LePrEF in the size and compression rates of search engine indexes (2) Its impact in query processing times. For the former, we show both the impacts of the different approaches in the final size and number of bits per entry of the two datasets adopted. For the latter, we show the results of processing sets of 5000 conjunctive queries. Conjunctive queries were chosen for being the most common type of queries submitted for search engines (de Moura et al., 2008, Saraiva et al., 2001).

The datasets adopted are GOV2 and WBR10. GOV2 is a Web collection comprised of 25,205,179 documents selected from a crawl of the .gov domain. We chose to use GOV2 in our experiments because the LETOR MQ2007 dataset, in which we performed our quality experiments, is a subset of GOV2, and thus it is important to include it also in the performance experiments. We used 5000 from the million query track (topics 1-5000) to measure the performance of LePrEF and the baselines in the GOV2 collection.

In order to evaluate the impact of LePrEF in a larger dataset, we also present its performance results when using the WBR10 collection. The WBR10 collection is a crawl of the Brazilian web that contains 127 million HTML documents, with about 3.6 terabytes, 6.8 billion links between its pages,7 gigabytes of text in the title, 100 gigabytes of anchor text and 510 gigabytes of body text. We also have access to a log of 1,203,024 distinct queries submitted to a Brazilian web search engine, from which we randomly selected a set of 5,000 queries.

WBR10 collection was created in our research group and will be publicly available for other research groups. Besides the difference in size in comparison to GOV2, WBR10 also provides a quite comprehensive crawling of the Brazilian web. It presents an abundance of links and anchor text information between its pages, thus providing a scenario that is closer to the one expected in a large scale search engine.

The query processor adopted in the experiments uses a document-at-a-time strategy, with the deployment of skip-lists in order to reduce the cost for processing queries, and uses compression methods to reduce the index size. More details about such techniques can be found in (Zobel and Moffat, 2006). The index is compressed using Elias-δ (Elias, 1975) for compressing the frequencies and UTI values in the index.

Skip-lists are structures that accelerate the query processing in document at a time query processing systems. They consist of indexes pointing to blocks of the inverted list and allow fast access to information about the block where each document can be found. In our experiments, we create an entry in the skip-lists for each 1000 documents of each inverted list. This number was chosen through experiments. In the case of the frequencies, the values stored were the $1 + log_2$ of the frequency, a common option representation adopted in search systems (Baeza-Yates and

Ribeiro-Neto, 1999).

As the learn to rank baseline for the performance experiments, we chose to use Genetic Programming (GP). Besides yielding very strong retrieval quality results, its simple and straightforward implementation leads to low computational costs in comparison to the other Learn to Rank approaches. Furthermore, we decided to use as baselines two possible implementations, which differ according to the way they store the multiple features in the index:

a) **GP with Feature Vector.** Every entry in the inverted list contains, for a term-document pair, its frequencies for all text based features. This method has the advantage of increasing the locality of the data, since no further seeks in the dataset are necessary to fetch values for all the features, but has the disadvantage of increasing the index size, since it becomes necessary to indicate when an entry has frequency zero for any feature.

b) **GP over Top Results + Fetch.** The top 1000 results for each feature are fetched, and their results are joined into a single resulting list. As some documents in this resulting list may not have all their feature information fetched in the first phase, their complementary feature values are fetched in a second phase, so that all the results in this joined list have all their feature values retrieved.

Since our goal is to illustrate the possible differences in costs that arise when using LePrEF, we evaluated the performance of LePrEF in comparison with the baseline by submitting the selected queries to our previously described search engine, using a simplified query processing. This procedure allowed us to have a better control of the experiments.

The experiments were run in a machine with 16 GB of RAM, CPU AMD Opteron series 2376, 2.3 GHz, 512 KB L1 cache, quad-core and using a single SATA II disk with 2TB. To evaluate the performance of the systems, we adopted the software Httperf (Mosberger and Jin,

1998) to generate the workload to all query processors evaluated. It is a software that is intended to measure the performance of a web server from the client perspective, by evaluating, among other metrics, average response time for the client to receive its answer and query throughput, which are the results that we will present.

3.5.2 Efficiency Results

In Section 3.4.2, we presented the average number of bits per compressed entry in the inverted list of the LETOR dataset, and concluded that the chosen formula for Fold 5 leads to the best results in that scenario. In order to show if that conclusion holds true when considering the whole GOV2 and the WBR10 collection, we present the overall size of the inverted index and the average number of bits per entry (only the frequency/UTI value is considered in this average; document IDs are not considered) after the indexing for both collections. These results are presented in Table 3.3

Table 3.3: Size of indexed datasets, both in overall size and average number of bits per frequency/UTI entry

Source Index	WBR10		GOV2	
	Size(Gb)	Bits /Entry	Size(Gb)	Bits /Entry
Text	44	1.98	7.4	1.78
Title	1.5	4.8	0.2	3.97
Anchor	0.8	6.99	0.21	6.95
URL	0.4	5.26	0.017	2.61
Feature Vec.	63	7.24	12	7.20
Fold 1	58	5.72	10.5	6.36
Fold 2	59	5.99	12	7.63
Fold 3	52	4.25	9.7	5.14
Fold 4	67	8.26	12	8.51
Fold 5	42	1.32	7.4	1.79

At first glance, it can be seen that the best formula when considering the number of bits per entry in the LETOR Dataset was also the best of the five folds, fold 5. It is also interesting to notice that, while in GOV2 the final size of the UTI index has about the same size of text index, in WBR10 the UTI index was even smaller than the text index. This property in WBR10 is due mainly to the fact that many entries that have frequency values higher than 1 in the text index, have respective UTI values set to 1 by LePrEF, thus requiring less bits to be represented.

Another interesting result to contemplate is the fact that, while fold 2 was the worst performing UTI formula bits per entry-wise in LETOR, in GOV2 and WBR10 it was the second worst performer, behind fold 4. This behavior was mainly caused by differences in the distribution of values between functions of fold 2 and fold 4. The function for fold 4 assigned higher values to the entries at the bottom of the distribution, thus assigning higher values to entries considered less important. While the LETOR collection is composed of the top selected entries associated with each query, being thus a selected set of more relevant entries, the other two collections are complete, which explains such differences.

We can also see that the option of a single index containing the values for all features in each entry in the inverted list lead to a much larger dataset. This result was somewhat expected, since there is a need for an extra number of bits to identify whether the value of a feature is present or not in each entry.

These results show that it is important to take into consideration not only the quality of results, but also the distribution of UTI values generated by the function chosen by LePrEF to generate the final index. The choice of the function has a direct impact on compression rates and we pretend to further study this topic in future works.

Figure 3.8(a) presents the average response time in each query processor, as the number of requests per second increases for the GOV2 collection. At request rates up to 25 per second, only a slight difference can be verified between using LePrEF or more traditional approaches, with GP using feature vectors achieving the worst performance (250 milliseconds, against 108 from LePrEF and 129 from GP+Fetch) in these smaller rates. However, when the number of submitted queries is above 30, GP with fetching performance has a huge increase in response time, with average time raising to 4972 milliseconds, probably associated with the much larger number of disk seeks performed in this approach. Using GP with feature vectors also started to show signs of saturation, but still with a much smaller response time of 308 milliseconds, and keeps competitive response times until 32 requests per second, where its response time raised to 3433 milliseconds.

On the other hand, when using LePrEF, the response times keep a steady response time until the request rate is at 46 requests per second, where the response time was 131 milliseconds, and increasing to 3900 milliseconds with a rate of 47 queries per second. These results are a strong indication that, performance-wise, indeed the deployment of LePrEF can lead to massive efficiency gains in terms of response time.

As expected from the method with the best response times, when considering the query throughput with GOV2, once again LePrEF had an outstanding performance in comparison to the baselines. Figure 3.8(b) presents their results in this scenario. As can be seen, up to 29 requests per second, the query throughput matches closely the number of requests made by second. The worse performer, once again, was GP+fetching new evidences, with a maximum throughput of 29 queries per second. GP+feature vector had a better result, with a throughput of 31 queries per second. LePrEF, on the other hand, had a much superior performance, with throughput up to 46 queries per second.

In order to evaluate how the proposed method fares in relation to the baselines in a larger dataset, we also present experiments done with the WBR10 collection. In Figure 3.8(c), we present the average response times of the three approaches in WBR10. As can be seen at first glance, the overall response times when processing queries in WBR10 were much higher than in

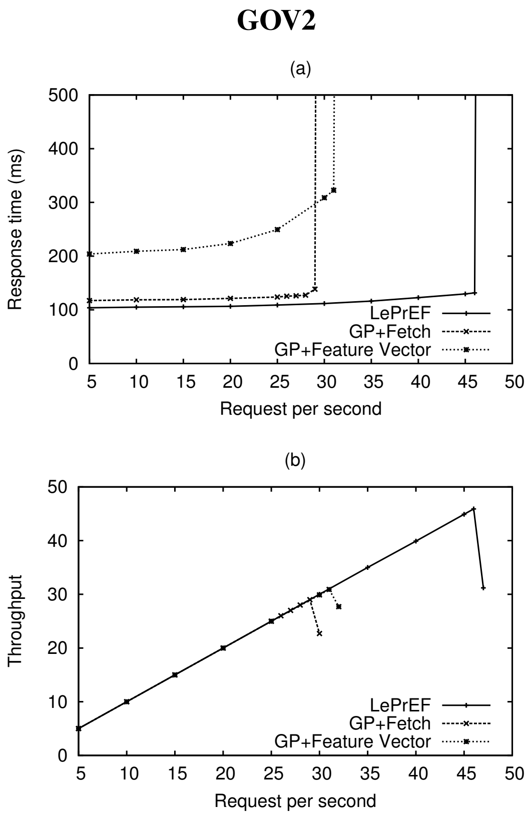

Figure 3.8: Response times and saturation levels for LePrEF and the two GP variations for GOV2 and WBR10

GOV2, which was expected due to its much larger size. An aspect of these results that differs from the results obtained with GOV2 is the response times of GP+Fetching and GP+Feature Vector, with GP+Fetching achieving overall smaller response times in comparison to GP+Feature. GP+Feature vector holds well until a rate of 3 queries per second, with an average response time of 1.08 seconds, while with GP+Fetching the best rate/time ratio was with 6 queries per second, achieving an average response time of 1.60 seconds, while LePrEF once again obtained the best results, with its best results at 9 queries per second, and average time 1.2 seconds.

In Figure 3.8(d), the query throughput of the three approaches is presented. Once again, LePrEF was visibly superior to the baselines. GP+Feature Vector obtained the worst result, with a maximum query throughput of only 5 queries per second, while GP+Fetch was slightly better, with 7 queries per second of maximum query throughput. On the other hand, LePrEF outperformed both baselines, with a maximum query throughput of 11 queries per second, 57% better than the next best alternative.

It is important to stress that these results were achieved in a simulated setting. In real search engines, the query processing is a much more convoluted process, and in the other hand, various approaches are used in order to further minimize the computational costs of this process, many times with a quality trade-off. Nevertheless, the results presented represent an evidence that deploying LePrEF and the concept of UTIs in a search engine architecture can lead to significant efficiency gains, and must be further investigated regarding other aspects, such as possible impact in pruning techniques and how to use it with sources of relevance evidence that are not available at query indexing time.

Chapter 4

Finding Phrasal Terms in Text Datasets

Since LePrEF is essentially a bag-of-words based approach, we also investigated techniques to add word relationships in IR models, in order to possibly achieve higher precision levels. In this chapter, we show our efforts in automatically detecting phrasal terms and using them to improve information retrieval tasks. We consider a *phrasal term* as a sequence of words that has a function of phrasal verb, phrasal noun, phrasal adjective or phrasal adverb in a text. Such phrasal terms are sequences of words that have a specific meaning, which can even be different from that of the individual words that compose the phrasal term. Examples of phrasal terms are “artificial intelligence”, “comic book”, “tax free”, “legal issues”, “formula one”, “New York”, and so on.

The task of finding phrasal terms may be associated with several applications related to text processing, such as text classification, search and keyword finding (tau Yih et al., 2006). The objective of identifying and modeling phrasal terms is to produce a better representation of text documents when compared to the processing of only individual words as units. Individual words can, for instance, be composed together to build expressions with completely different meanings and such information may be missed when not modeling compositions. A prior knowledge of which word sequences have a specific meaning can be used for many practical purposes when processing textual sources of information, enriching the representation of the content found on those sources.

As a sample of application of phrasal term detection methods, we can mention the enrichment of traditional information retrieval models. Most popular textual information retrieval models represent documents as "bags of words", i.e., sets of completely independent distinct terms (Baeza-Yates and Ribeiro-Neto, 1999), disregarding the order in which the words appear in the document. The extraction of these independent words is done by the use of text parsers, which usually separate these terms in strings of alphanumeric characters separated by non-alphanumeric characters. After the parsing, each term is considered as an independent piece of information present in the document, with total disregard to its placing and the terms around it.

While this approach has yielded satisfactory results, it is clear that considering the words in a document as independent is not a faithful representation of the actual relationship between them. Words can be composed together to build expressions with completely different meanings (e.g., "artificial intelligence", or "comic book"). Considering only individual words as terms can lead to loss of information, which can mislead the system and lead to erroneous results due to ambiguity. For instance, the word "gore" in a query might mean that the user is interested in horror movies; however if in the query the word "gore" is preceded by the word "al", it probably means that the user is interested in the politician Al Gore.

Here we argue that a prior knowledge of which word sequences have a specific meaning is important and can be used for many practical purposes in Information Retrieval, enriching the information available about documents. This is particularly important in web search applications, where the precision of results is more important than the recall; and where the most common type of queries, the informational queries, usually have larger numbers of words (Baeza-Yates et al., 2006). This property raises the chance of informational queries containing phrasal terms. Further, informational queries rely mostly on the textual content as a source of information to improve the quality of search results, thus a better representation of texts in the information retrieval model may have higher impact in these web search engine queries.

Word sequences and word co-occurrence statistics have been shown useful in many different information retrieval tasks (Tan et al., 2002, tau Yih et al., 2006, Tesar et al., 2006, Zhang et al., 2007). These examples of previous efforts to deal with word sequences in different information retrieval tasks represent a strong indication that indeed the detection and employment of meaningful word sequences can lead to improvements in the quality of various information retrieval tasks.

Our solution to detect phrasal terms uses statistical features of the word *n-grams* (sequences of n words) found in the documents and a classifier to determine which are valid phrasal terms. We present empirical evidence showing that this is an effective solution to this problem. The intuition behind this approach is that phrasal terms share similar statistical features that might be useful to distinguish them from non-phrasal sequences. It provides an effective mean of discovering meaningful word sequences that is grammar independent, since it is not explicitly dependent on grammatical rules, and time and space efficient, since it does not require the use of complex Natural Language Processing methods. Experiments indicate that, as the database grows larger, the effectiveness of the proposed method also increases, achieving precision values up to 94,45%.

We also show that using these phrasal terms can indeed lead to better results in information retrieval tasks. To this effect, we propose two different approaches that are able to use this information to enhance results in document search tasks, and perform experiments demonstrating the usefulness of our approach. Experiments in four different collections show gains of up to

36% in Mean Average Precision, when compared to a traditional single-word search.

4.1 Detecting Phrasal Terms

We define a *phrasal term* as a sequence of words that has a function of a phrasal verb, phrasal noun, phrasal adjective or phrasal adverb in a text. It is important to notice that, in our definition of phrasal term, it is crucial that the set of words has a specific, understandable, meaning. For instance, while “Carnegie Mellon University” is a phrasal term, “Mellon University” is not since it loses its meaning without the term “Carnegie”.

Our detection method, based on machine learning, uses statistical features of the word bigrams found in the documents to classify each of them as phrasal terms or not, since we believe that phrasal terms share similar statistical features that might distinguish them from non-phrasal sequences. We believe that this happens because the co-occurrence of words in a language is governed by a fixed set of rules—the grammar. However, it is important to stress that the proposed approach is not based on the grammar itself, but in examples of phrasal terms and non-phrasal sequences.

In the experiments presented, we focus on the detection of two-word phrasal terms. It can, nevertheless, be easily expandable to n-word sequences. Also, we ignored n-grams that contained numerical characters and stop-words, in order to prune the number of n-grams considered. Preliminary experiments done without the exclusion of these bigrams showed that they introduce noise into the classification, with no distinguishable advantage over their exclusion.

The proposed method can be divided into three main stages, as illustrated in Figure 4.1. In the first stage, we extract all existing bigrams from the document collection and compute their

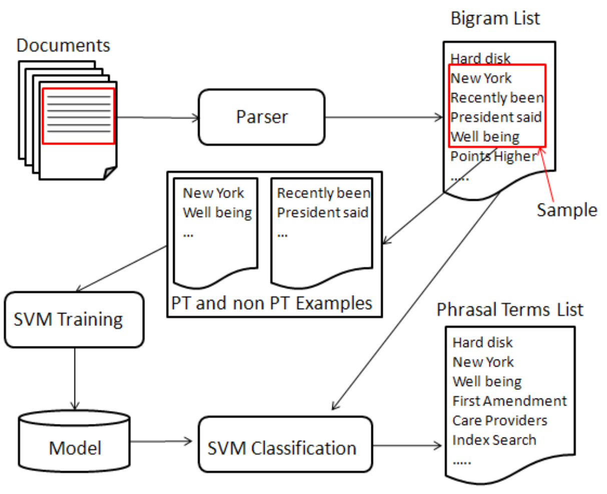

Figure 4.1: The phrasal term (PT) detection process.

features. To do so, the method parses all documents in the collection, collecting statistics about the individual terms and about the bigrams. This stage has a computational cost linear to the number of documents, and it is important to note that this stage can be trivially implemented during the indexing of the collection.

In the second stage, we manually label a small selected subset of bigrams as phrasal terms or non-phrasal sequences. This subset can then be used as training data for a Support Vector Machine (SVM) classifier.

Finally, in the third stage, we use the SVM classifier to process all bigrams and determine which are proper phrasal terms. It is important to stress that this classification is fairly inexpensive, especially because the method uses a reduced number of features.

4.1.1 Identifying Phrasal Term Candidates

As we mentioned, we have worked in this thesis with only phrasal terms composed of two individual words. At first glance, identifying all phrasal terms existing within all possible bigrams of text in a text collection can be seen as a very complex operation. In fact, if we consider all possible two-word sequences, the maximum number of candidates is $|V|^2$, where $|V|$ is the size of the vocabulary. However, in real scenarios, the majority of possible sequences never actually occur, largely reducing this number. A simple algorithm, linear in the size of the collection, to find all bigrams present in a collection is:

a) For each document in the collection.

    1. Parse the document and extract every sequence of two words and its statistics.

    2. Whenever main memory is full, dump all collected sequences to secondary storage.

b) Join all runs of sequences produced into a single list of candidates.

As stated before, in this step the method ignores bigrams with stop-words and with numeric characters, which reduces the number of bigrams to be processed.

4.1.2 Statistical Features

We argue that only a basic set of word occurrence and co-occurrence statistics is needed to determine whether a bigram is a meaningful phrasal term or not. Even though no language-dependent grammar or syntactic information is used, we believe such statistics carry enough information to provide an accurate description of what is and what is not a phrasal term.

The features we adopt here are mostly based on frequency counting, and have a straightforward calculation, adding little computational cost to the feature extraction. Given an *ordered pair of words* (t_1, t_2), we propose the following set of features:

a) **Pair probability** ($P(t_1, t_2)$): The number of times pair (t_1, t_2) occurs in the collection, over the number of pairs in the collection. A large frequency may indicate that a bigram is a possible phrasal term. On the other hand, a small frequency is not necessarily an indication that the bigram is not a phrasal term.

b) **Pair Probability given** t_1 **(**$P((t_1, t_2)|t_1)$**)**: The number of times pair (t_1, t_2) occurs, over the number of times that t_1 occurred. This feature indicates how likely it is that the presence of t_1 is an indication that the next word is t_2. We argue that t_1 being often followed by t_2 is an indication that the pair is a phrasal term.

c) **Pair Probability given** t_2 **(**$P((t_1, t_2)|t_2)$**)**: The number of times pair (t_1, t_2) occurs, over the number of pairs that end with t_2. Similar to the Pair Probability given t_1 feature.

d) **Document Probability of** (t_1, t_2) **(**$P_D((t_1, t_2))$**)** : The number of documents where pair (t_1, t_2) occurs, over the total number of documents. Notice that this is different from Pair Probability, since it relates to the number of documents where the pair appears, instead of its absolute number of occurrences.

e) **Document Probability of** t_1 **(**$P_D(t_1)$**)**: The number of documents where word t_1 occurs, over the total number of documents. This feature should reflect the relative rarity importance of word t_1, in a role similar to that of the inverse document frequency (IDF), traditionally used in the Vector Space Model (Baeza-Yates and Ribeiro-Neto, 1999).

f) **Document Probability of** t_2 **(**$P_D(t_2)$**)**: The number of documents where word t_2 occurs, over the total number of documents.

g) **Raw Pair Probability given** t_1 **(**$P_R(t_1)$**)**: The number of distinct pairs starting with t_1, over the number of distinct pairs.

h) **Raw Pair Probability given** t_2 **(**$P_R(t_2)$**)**: The number of distinct pairs ending with t_2, over the number distinct of pairs.

It is important to notice that, while the proposed set of features is fairly simple, the experiments presented in Section 4.3 show that these features are good enough to allow for a satisfactory precision in the phrasal terms detection task. The importance of each feature is studied in Section 4.3.2.

After all the bigrams and their statistics are extracted, examples of phrasal terms and non-phrasal sequences are manually extracted from a random sample of bigrams, and used to train an SVM classifier model. The remaining feature vectors for all the bigrams are applied to this classifier model, which will determine whether each bigram is a meaningful phrasal term or not. As a result of these steps, a list of detected phrasal terms is created.

We note that, while in this work we use an SVM classifier to select phrasal terms, other classification techniques could be used.

In the following section, we describe how these discovered phrasal terms can be used for enhancing search in text collections.

4.2 Using Phrasal Terms

After the detection of phrasal terms, it is important to study how this information can be used in Information Retrieval tasks. In this thesis, we study the particular case of text document searching. We chose to evaluate the impact of phrasal terms using the traditional Vector Space Model (VSM) with TF-IDF term weighting (Baeza-Yates and Ribeiro-Neto, 1999) and propose two different approaches to apply phrasal terms in a search task:

a) **Phrasing:** Phrasing consists in parsing the user query in order to find phrasal terms. Whenever one is found, it is treated by the search engine as a *phrasal query*. Phrasal queries are queries in which, for a document to be considered relevant, it is mandatory that it possesses the designated phrase. The phrases are usually represented by the use of quotes. For instance, if our method detected “Los Angeles” as a phrasal term, the query ‘*Los Angeles Hotel*’ would be processed by the search engine as the query ‘*“Los Angeles” Hotel*’, with the search engine considering “Los Angeles” as a phrase.

b) **Expansion with Phrasal Terms:** In this case, all the detected phrasal terms in the collection are treated as single-word terms, and indexed by the search engine as such. All individual words are still indexed. In practice, we are expanding the documents in the collection to contain, besides all its individual words, also all its phrasal terms. All document norms are recomputed taking this into account. The same process is applied to the query. For instance, in a scenario where the proposed approach detected “Los Angeles” as being a phrasal term, the query ‘*Los Angeles Hotel*’ would be expanded into a query with four distinct terms: “*Los*”, “*Angeles*”, “*Hotel*”, and “*Los Angeles*”. The same would apply to all documents.

4.3 Experiments

The experiments are divided into two main parts: phrasal term detection and search impact evaluation. Before presenting them, we detail the experimental setup.

4.3.1 Experimental Setup

In the experiments performed, we use four datasets: FBIS, Wt10G, L.A. Times and OHSUMED. We chose these four datasets since they contain documents of different natures, topics and sizes, which allowed us to examine the behavior of our method in distinct scenarios. Furthermore, all these datasets have a previously evaluated set of queries readily available, which allowed us to use them in the search experiments.

The Wt10G collection (Soboroff, 2002) is comprised of 1,692,097 documents selected from a crawl of the World Wide Web. This dataset is useful to show how the methods behave in a free-publication environment, such as the Web. While Wt10G may be considered small when compared to true web search scenarios, previous studies have shown that this dataset, despite being only a sample, retains many of the properties of larger web crawls (Soboroff, 2002).

The OHSUMED document collection (Hersh et al., 1994) is a subset of MEDLINE, containing 348,566 medical articles. For this work, we indexed only the title and abstract of the documents.

The FBIS dataset is part of the TREC document collection (disk 5) (Voorhees, 1999). It contains 130,471 randomly selected newspaper articles. This collection, while having fewer documents than OHSUMED, actually contains more text, since it contains full newspaper articles, regarding news from all around the world.

Finally, the L.A. Times collection is also a part of the TREC document collection (disk 5) (Voorhees, 1999). It contains about 131,896 randomly selected newspaper articles, published between 1989 and 1990 in the Los Angeles Times newspaper. Unlike FBIS, all articles are taken from a single newspaper, hence the collection has a more limited vocabulary and set of topics.

Table 4.1 shows some statistics about the databases. We can clearly see that, while containing more documents than L.A.Times and FBIS, OHSUMED has a smaller vocabulary and a smaller number of bigrams. It is also important to notice that the removal of bigrams with numbers and with stop-words had a huge impact in the number of bigrams, removing about 50% of the bigrams present in Wt10G.

	Voc. Size	Total Bigrams	Considered
OHSUMED	164407	5590058	2971596
FBIS	400073	7595417	3842587
L.A.Times	248660	9440633	5518509
Wt10G	5646913	91143120	47885424

Table 4.1: Statistics about the text collections utilized in the experiments. Total Bigrams is the absolute number of bigrams in the database. Considered is the number of bigrams without stop-words and numeric characters.

Classification Setup

As described in Section 4.1, in this work we used an SVM classifier to discover sets of phrasal terms from a set of bigrams. The SVM implementation used was *libsvm* (Chang and Lin, 2001). The kernel adopted was the RBF kernel, since in preliminary experiments it consistently outperformed other kernels such as polynomial and sigmoid, albeit not by a large margin.The training parameters were optimized using scripts included in libsvm.

Bigrams with frequency values smaller than 10 were discarded as phrasal term candidates, since they contain too little co-occurrence information. Also, since these bigrams are so infrequent, the possible gain yielded by detecting phrasal terms among them may be small, in comparison with the overhead necessary to classify them all.

For each database used in this experiment, 500 phrasal terms and 500 non-phrasal sequences were randomly sampled from the bigrams found in the databases, to be used as training and testing sets. The training set was stratified (Witten and Frank, 2000) in this manner because the set of bigrams that is not a phrasal term is much larger than the set of phrasal term bigrams, and thus a purely random sample would also have a disproportional number of non-phrasal terms, which could bias the classifier, and preliminary experiments done with an unstratified set of examples led to a classifier with overall much worse results. All tests were performed using 10-fold cross validation. The precision and recall values obtained represent the average of the

10 runs. While in the experiments presented we used 1000 examples in the construction of our classification model, in all databases when using as few as 100 examples to construct the model our method yielded similar precision and recall values as using all the 1000 examples, as shown in Figure 4.3 in section 4.3.2.

In the case of OHSUMED, which is comprised mostly of medical terms, technical expertise was needed in order to identify the phrasal and non-phrasal terms. Thus, to do so we asked for a Pharmacist with a Masters degree in Tropical Pathologies to do this identification.

To evaluate the phrasal term detection, we used the traditional classification metrics of micro-averaged precision, macro-averaged precision, macro-averaged recall and macro-averaged $F1$ (Witten and Frank, 2000). We also present the total precision and recall for the phrasal terms class and non-phrasal sequences class. These measures are defined as follows:

Let C be the set of bigrams used for testing. We define $C_{phrasal} \subseteq C$ as the set of phrasal terms within C and $C_{not} \subseteq C$ as the set of non-phrasal sequences within C. Let $C'_{phrasal}$ be the set of bigrams classified as phrasal terms by the SVM classifier, and let C'_{not} be the set of bigrams classified as non-phrasal sequences by the SVM classifier.

The precision of the SVM classifier for a given class $c \in \{phrasal, not\}$ is the percentage of bigrams correctly classified as being of class c, over all bigrams classified as being of class c:

$$Pr_c = \frac{|C'_c \cap C_c|}{|C'_c|} \tag{4.1}$$

Recall is defined as the percentage of bigrams correctly classified as being of class c, over all bigrams that actually belong to class c:

$$Rc_c = \frac{|C'_c \cap C_c|}{|C_c|} \tag{4.2}$$

Micro-averaged precision is defined as the percentage of correctly classified candidates within the set of all classified candidates:

$$\mu Pr = \frac{|C'_{phrasal} \cap C_{phrasal}| + |C'_{not} \cap C_{not}|}{|C|} \tag{4.3}$$

Macro-averaged precision is defined as the average precision for all classes:

$$Pr = \frac{Pr_{phrasal} + Pr_{not}}{2} \tag{4.4}$$

Macro-averaged recall is defined as the average recall for all classes:

$$Rc = \frac{Rc_{phrasal} + Rc_{not}}{2} \tag{4.5}$$

Macro-averaged $F1$ is the harmonic mean between macro-averaged recall and macro-averaged precision:

$$F1 = \frac{2PrRc}{Pr + Rc} \tag{4.6}$$

Document Searching Setup

Search results for both the original datasets and after the inclusion of the detected phrasal terms were evaluated in terms of Mean Average Precision (MAP) and precision at the top 10 results provided for each query (Pg@10) (Baeza-Yates and Ribeiro-Neto, 1999). Topic queries used in our experiments are from 1 to 106 of OHSUMED, from 351 to 400 of FBIS, from 401 to 450 of L.A. Times, from 451 to 500 of Wt10G, where only the titles portion of the queries were considered.

4.3.2 Phrasal term detection

Table 4.2 shows the phrasal term detection results achieved by our method in the four databases. In general, results show that our method is a robust solution to the phrasal term detection problem. In terms of precision, in all collections, satisfactory results were obtained.

	OHSUMED	FBIS	L.A.Times	Wt10G
μPr	70,80%	76,10%	81,90%	94,40%
$Pr_{phrasal}$	70,72%	77,94%	85,05%	95,87%
Pr_{Not}	70,88%	74,48%	79,27%	93,02%
Rc_{Comp}	71,00%	72,80%	77,40%	92,80%
Rc_{Not}	70,60%	79,40%	86,40%	96,00%
Mac_{Prec}	70,80%	76,21%	82,16%	94,45%
Mac_{Rec}	70,80%	76,10%	81,90%	94,40%
Mac_{F1}	70,80%	76,16%	82,03%	94,42%

Table 4.2: Results of phrasal term detection in the Wt10G, FBIS, L.A.Times, and OHSUMED datasets.

The lowest values obtained were for the OHSUMED collection. Results, in all the metrics, were around 71%. By analyzing the data, we can conclude that the size of the documents had an important role in this behavior. Documents in the OHSUMED database are fairly small, containing only titles and abstracts, which reduces co-occurrence information available for the bigrams. However, as can be seen on Section 4.3.3, even with relatively small precision values for OHSUMED, gains were still achieved when using the detected phrasal terms to search documents.

In the remaining databases, we observed that precision for class “phrasal terms” is higher than for class “non-phrasal sequences”. The SVM classifier appears, therefore, to be conservative when choosing phrasal terms. This could, of course, be changed by further tuning the classifier. However, this is not the focus of our work. Precision values were around 71%, 78%, 85%, and 95% for the OHSUMED, FBIS, L.A. Times, and Wt10G collections, respectively.

These results, even with a difference of about 24% when comparing different databases,

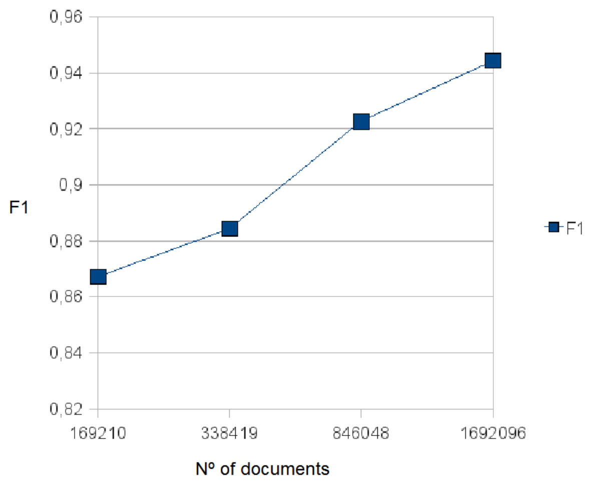

Figure 4.2: F1 scores obtained when applying the phrasal term detection method to portions of different sizes of the Wt10G Collection.

show that the method is a simple and good alternative to phrasal term detection. However, it is important to verify why the results in the different databases had that much difference. The F1 values obtained in WT10G are on par with the results shown in (Zhang et al., 2007), that were around 90% of F1. However, a direct comparison is not possible since they used a different dataset.

These results may indicate that the size of the database might have a strong influence in the overall quality of the classifier, since as the size of the database grew, so did the quality metrics values. In order to investigate whether the assumption that the size of the database effects the phrasal term detection holds true, we repeated the experiments using the Wt10G collection and splitting it into portions of different sizes: 1/2 of the Wt10G database, 1/5 of the Wt10G and 1/10 of the Wt10G database. The examples used in these experiments were mostly the same as in the previous experiment, with the exception of the examples that had a frequency smaller than 10 in that slice of the database, which were replaced by new randomly sampled examples. The results are presented in Figure 4.2.

These results confirm that the size of the database indeed influences the phrasal term detection.

It is also important to notice that as the database grows, the impacted caused by the addition of new pages diminishes, which is an indication that after a certain number of pages, using more pages for the phrasal term detection will probably have almost no effect in the precision of the detection procedure.

At first glance this does not hold true when considering the results for the FBIS collection and the L.A. Times collection, since they have about the same number of documents and similar sizes, yet different detection performance. However, the FBIS database is comprised of articles with a much broader spectrum of topics in comparison to L.A. Times, since FBIS is a translation of foreign texts regarding many different countries while L.A. Times is composed of news articles that are pertinent to the Los Angeles area citizens, having, thus, a much more tight domain. This broader spectrum of FBIS leads to a larger vocabulary size, as shown in Table 4.1.

	# of bigrams	# of Phrasal Terms
OHSUMED	141724	38687
FBIS	163143	62451
L.A.Times	160370	34013
Wt10G	3038832	69180

Table 4.3: Total of bigrams considered (Frequency larger than 10 and no stop words) and total detected as phrasal terms by our method.

Table 4.3 shows the number of bigrams found and considered in each dataset (i.e. bigrams with frequency higher than 10 and without stop words), and the number of bigrams classified as phrasal terms by our method. However it is important to notice that these are the bigrams that our method decided are phrasal terms, and not necessarily actual phrasal terms.

By examining these results, it is interesting to notice that, as expected for being the largest, Wt10G yielded the largest number of bigrams. However, this large number of bigrams did not translate into a much larger number of phrasal terms in comparison with the other databases, specially FBIS. Notice also that there is no obvious correlation between the number of bigrams and the number of phrasal terms detected.

Feature Impact study

In this section, we study the impact that each feature has on the phrasal term detection task. To do so, we first calculate the information gain of each feature, in order to verify which features are the best class discriminators in each database. Afterwards, we studied the impact of each feature in the final classification result, and analyzed the impact of removing several combinations of the less discriminative features from the classification in each database. For a better visualization, the names of the features in this section will be replaced by numbers, given by Table 4.4.

Id	Feature	Id	Feature
1	$P(t_1, t_2)$	5	$P_D((t_1))$
2	$P((t_1, t_2)\|t_1)$	6	$P_D((t_2))$
3	$P((t_1, t_2)\|t_2)$	7	$P_R(t_1)$
4	$P_D((t_1, t_2))$	8	$P_R(t_2)$

Table 4.4: Identifiers for each feature in the following tables.

Wt10G		L.A.Times		FBIS		Ohsumed	
Id	InfGain	Id	InfGain	Id	InfGain	Id	InfGain
1	0.6945	1	0.4066	2	0.2138	2	0.0824
4	0.6372	4	0.3665	1	0.2012	5	0.0707
2	0.4864	2	0.2509	4	0.1636	7	0.0564
3	0.4609	3	0.2281	3	0.112	3	0.0461
6	0.0178	8	0.0691	7	0.0481	6	0.0439
8	0	6	0.041	8	0.0443	8	0.0393
5	0	7	0.0227	5	0.0439	4	0.0243
7	0	5	0	6	0	1	0.0147

Table 4.5: Information gain of each feature in all databases. The features are identified according to the Ids in Table 4.4

Table 4.5 shows the information gain of each feature for each database. A first observation is that feature 1(pair frequency) is among the best features in most cases, except for OHSUMED collection. As it can be seen, in OHSUMED all features had small information gain values, which explains the worse classification results in this collection.

Table 4.5 also shows that, in Wt10G, L.A. Times and FBIS, features 5, 6, 7 and 8 (Document probability of t_1 and t_2 and Raw probability of t_1 and t_2, respectively) yielded the smaller information gain values. This is an interesting and somewhat expected result, since these features regard information from single word occurrences (document probability of the words and number of bigrams the word is on) and not from the bigrams. Nonetheless, we believe that, even though these features are not good individual discriminators, their use in addition to other features may indeed lead to a positive impact in the results, especially in smaller databases where the bigram informations have a smaller information gain value.

To assert whether this belief is accurate, we performed the same experiments of Section 4.3.2, but removing all the combinations of the features 5,6,7 and 8. Table 4.6 shows the results for some of these combinations in terms of F1 when removing the feature sets $\{5,6,7,8\}$, $\{5,6\}$ and $\{7,8\}$. Other combinations were omitted because they lead to similar conclusions. As it can be seen, while in L.A. Times and FBIS databases the use of these features indeed translated into a higher F1 value, this does not hold true for Wt10G, where removing these features had little impact on the final F1 result. This may be an indication that, since Wt10G is much larger than the other databases, the bigram statistics are enough to allow for a precise classification in Wt10G.

	- 5,6,7,8	- 5,6	- 7,8
OHSUMED	0.6947	0.7096	0.6953
FBIS	0.7536	0.7506	0.7576
L.A.Times	0.7982	0.8231	0.8202
Wt10G	0.9452	0.9462	0.9451

Table 4.6: F1 values for the classification task without feature sets $\{5,6,7,8\}$; $\{5,6\}$; $\{7,8\}$.

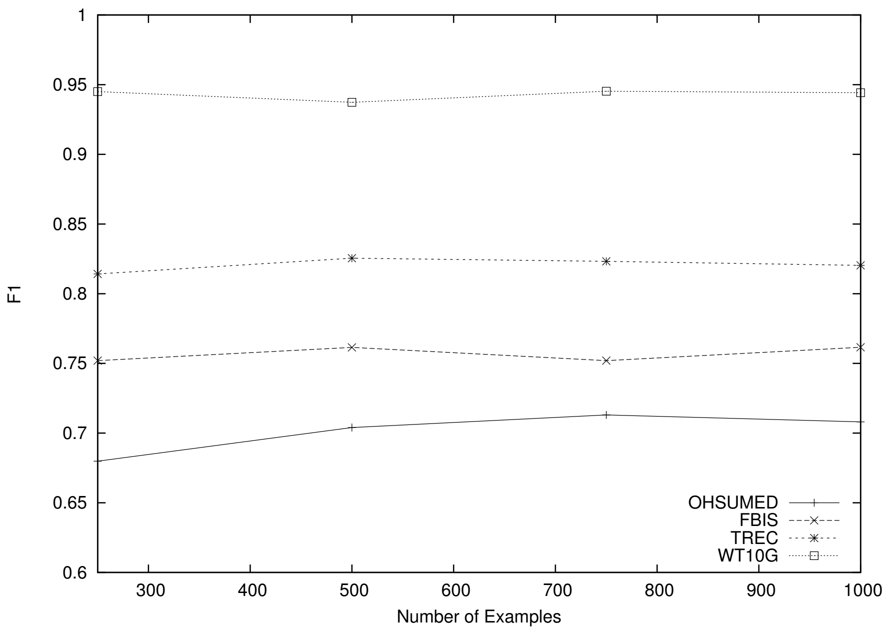

Figure 4.3: F1 scores obtained when varying the size of the training set to 250, 500, 750 and 1000 examples (divided half-to-half). Except for OHSUMD between 250 and 750 examples, the differences were not statistically significant (p-value¿ 0.05)

Training Set Size

In Figure 4.3, we present the F1 performance using different training set sizes (while still maintaining a 50% ratio for phrasal/non-phrasal terms) in all four datasets evaluated.

As can be seen 250 examples (125 phrasal and 125 non-phrasal) in all datasets but wt10g yielded worst results than using 500, which could be an indication that as the datasets grow larger, less examples might be needed. When comparing 500 to 750 examples, in both FBIS and L.A. Times there was a slight decrease in F1 while OHSUMED showed a small increase. However, when analyzing the big picture, for all datasets increasing the number of examples did not lead to a corresponding increase in quality of detection (while it could be argued that in OHSUMED until 750 examples there was a visible increase in quality, it can be stated that the onset of its best classification quality needed more examples due to its sparse quantity of information per document)

4.3.3 Application to Search

In the above experiments, we have presented results that show our approach can achieve high precision levels when detecting phrasal terms, specially in larger collections. However, it is also important to show that these phrasal terms can indeed have a positive influence in the quality of information retrieval systems. In this section we present results obtained by the application of phrasal terms in document searching.

We start by applying our method to each complete document collection. We used 1000 example bigrams to train the SVM classifier. Table 4.3 shows the number of bigrams in each dataset (bigrams with frequency higher than 10 and without stop words), and the number of bigrams classified as phrasal terms by our method.

Following, we look for the detected phrasal terms in the queries used for testing, and submit these queries with the added phrasal terms to a Vector Space Model Search Engine. We also evaluate the results obtained when manually picking all phrasal terms from the queries. This last experiment is useful to verify the impact the method would have if all phrasal terms present in the queries were detected, and to give a better insight about how the proposed method impacts search in comparison with the best-case scenario, that is human classification.

Table 4.7 presents the total number of queries and the number of queries that have phrasal terms, either detected by the method or manually detected, for each database. It can be seen that the OHSUMED collection has the largest number of queries with phrasal terms in the queries (75). This is due to the fact that the OHSUMED queries are substantially longer and have many medical terms. It is also interesting to notice that when manually picking phrasal terms, less queries were found (73). This is due to the fact that some of the phrasal terms found by our method were not actual phrasal terms . Conversely, only 11 of the Wt10G queries contained phrasal terms. Even though this small number may have a small impact in the overall results, the gain obtained when considering only the modified queries is expressive, as reported in the following.

	number of queries		
	total	automatic	manual
Ohsumed	106	75	73
FBIS	50	14	24
L.A.Times	50	9	21
Wt10G	50	11	12

Table 4.7: Number of queries modified for each database

	MAP					
	Baseline	Manual	Expansion	Gain	Phrase	Gain
Ohsumed	0,1853	0.1864	0,1934	4%	0,1872	1%
FBIS	0,1019	0.1082	0,1084	**6%**	0,1017	0%
L.A.Times	0,1424	0.1676	0,1555	**9%**	0,1429	0%
Wt10G	0,1043	0,1144	0,1116	**11%**	0,1025	-0,02%

Table 4.8: MAP Results obtained when considering all queries.

	P10					
	Baseline	Manual	Expansion	Gain	Phrase	Gain
Ohsumed	0,2367	0.2333	0,2393	1%	0,2401	1%
FBIS	0,1364	0.1434	0,1491	**9%**	0,1365	0%
L.A.Times	0,1473	0.1727	0,1618	**10%**	0,1455	-1%
Wt10G	0,1327	1364	0,1418	7%	0,1291	0,03%

Table 4.9: P10 Results obtained when considering all queries.

Tables 4.8 and 4.9 shows the results for all test queries. Column "Baseline" refers to the results obtained using VSM. Columns "Phrase" and "Expansion" refer to the use of the detected phrasal terms as phrases in the queries and to the use of phrasal terms to expand documents, respectively. Column "Manual" refer to the results obtained when manually picking bigrams.

It is noticeable that using the phrasal terms as phrases in the queries yielded little to no gain (even small losses in some cases). This is due to the small number of relevant documents in each collection, many of which did not contain the phrase being queried. Conversely, using the detected phrasal terms to expand the documents led to gains in all the databases, both in terms of MAP and in P@10. An important remind here is that the expansion methods results in less costly query results computation when compared to the option of phrases.

It is interesting to notice that, as the quality of the phrasal term detection increases, the gain obtained by the use of phrasal terms also increases. While in OHSUMED the gain was only of 4% in terms of MAP and 1% in terms of P@10, in Wt10G the gains were more expressive (around 10%).

It is also interesting to notice that, in comparison to manually detecting phrasal terms, the results obtained by the automatic classification were similar, and even greater as in FBIS and OHSUMED. This is a good indication that the impact of the proposed method was near the optimal possible impact yielded by finding phrasal terms in search queries.

MAP					
	Baseline	Expansion	Gain	Phrase	Gain
Ohsumed	0,1916	0,2052	**7%**	0,1942	1%
FBIS	0,1457	0,1636	**12%**	0,1450	-1%
L.A.Times	0,1560	0,2119	**36%**	0,1585	2%
Wt10G	0,0383	0,0616	**23%**	0,0364	0%
P10					
	Baseline	Expansion	Gain	Phrase	Gain
Ohsumed	0,2558	0,2642	3%	0,2606	2%
FBIS	0,2013	0,2273	**13%**	0,2013	0%
L.A.Times	0,1616	0,2424	**50%**	0,1515	-6%
Wt10G	0,0606	0,1212	**17%**	0,0404	0%

Table 4.10: Results obtained when considering only queries with phrasal terms.

When considering only queries that contain phrasal terms, the results are quite impressive. We present in Table 4.10 values obtained when considering only queries that contain phrasal terms. As can be seen, the gains obtained were significant, ranging from 7% to 36% in terms of MAP. In terms of P@10, although there was only a small gain in OHSUMED, in the remaining databases the gain ranged from 13% to 50%. Once again, the use of phrasing yielded poor results when regarding only queries with phrasal terms. On the other hand, it is clear that the use of phrasal terms to expand documents in search tasks can lead to expressive gains. The results obtained in the Wt10G database may seen oddly small (ranging from 0.03 to 0.06), but this is due to the fact that a few of the queries that had detected phrasal terms did not yield any relevant result in the 100 first results, having thus MAP and P@10 values of 0.

It is also important to notice that, when expanding a document with the addition of phrasal terms, there is a small change, in VSM, in the document's norm, since the phrasal terms are added as extra terms in the representation of documents. The documents norm is increased, reflecting the addition of the phrasal term information to the documents. Thus, it is important to measure how this change affected the results. To do so, in Table 4.11 we show the results obtained by queries that do not contain phrasal terms. We do so because the results in these queries are only affected by the change in the norm of the documents. The results of using phrasal terms for phrasing are not shown because phrasing does not change the quantity of information present in the documents.

	MAP			P@10		
	Orig.	Exp.	Gain	Orig.	Exp.	Gain
Ohsumed	0,1703	0,1649	**-3%**	0,1906	0,1789	-6%
L.A.Times	0,1394	0,1431	**3%**	0,1441	0,1441	**0%**
FBIS	0,0898	0,0921	**3%**	0,1176	0,1257	**7%**
Wt10G	0,1502	0,1504	**0%**	0,1921	0,1921	**0%**

Table 4.11: Results obtained when considering only the queries that do not contain phrasal terms.

The results in Table 4.11 show that the addition of phrasal terms had a small impact in all databases in terms of MAP, with the gain (or loss) being at most 3%, while in terms of P@10 the impact was slightly higher in some databases. It is interesting to notice that while most databases had a slight increase or no change in terms of MAP and P@10, in OHSUMED, the addition of phrasal terms to the documents yielded losses in both metrics. This is probably due to the nature of OHSUMED: it is composed of only the abstract and title of medical papers, which we verified that resulted in small, phrasal terms heavy documents, in which every phrasal term added to a document have a large influence in its norm. Further, the individual terms present on the phrasal terms from this collection are already quite specific and present low ambiguity when compared to the other two collections. This could cause the loss in precision presented in Table 4.11. Nevertheless, these results indicate that, while the additional information had some impact on the results, the gain obtained by the methods was mostly due to the phrasal term existing in some of the queries, and not due to the information added to the documents.

These results present a strong indication that the method can lead to gains in search tasks. Further, the comparison with manually identifying phrasal terms show that these gains are near the best-case scenario for classifying phrasal terms.

Chapter 5

Phrasal Terms in Learning to Rank Environments

The experiments presented in Chapter 4 indicate that our method to detect phrasal terms results in better quality of results in a scenario of text search only, disregarding potential differences of the application of the method in a scenario with multiple sources of relevance evidence. One example of such scenario is Web search, where search engines use a myriad of sources to compute the final ranking, ranging from linkage information to user click data.

There are examples in literature where the quality gains observed in single sources of relevance evidence lead to no visible gains when this source is fused with a number of others. Different reasons may cause such phenomenon, such as overlap of the relevant results better scored in a single evidence with scores from other sources, or occurrence of results with abnormally high scores in other sources of evidence, which may avoid a change in another evidence modifying the list of top ranked documents. Since these effects cannot be ruled out *a priori*, in this chapter we study how the addition of phrasal terms information to textual features in a multi-feature evidence fusion scenario affects the final quality of the results obtained.

Thus, one of our goals in this Chapter is to assess whether including phrasal term information to a search task with a large number of sources of relevance evidence can yield quality gains, and if such gains are on par with those obtained in text-only search. We believe that this investigation can lead to a better understanding of the full impact that the detection of phrasal terms can have in I.R. tasks, and possible new research directions regarding them.

Another important aspect we investigate in this Chapter is whether the usage of phrasal terms may positively affect the results achieved by LePrEF. One of the disadvantages of using LePrEF to learn to rank is that it does not have access to the term compositions available at query processing times. Thus, when taking at indexing times information about important term compositions provided by the phrasal terms, LePrEF may become an even more competitive learn to rank method. We investigate this hypothesis in this Chapter.

To do so, we used the LETOR benchmark collection (Liu et al., 2007). We had two particular reasons for choosing so: First, LETOR is a well known, largely employed dataset when comparing evidence fusion techniques, and thus results obtained in this dataset can readily compared to a large number of other fusion methods. Second, the LETOR dataset is based on TREC GOV2 dataset, which we also have access to. This means that we can easily extract co-occurrence information from it. Third, we used it in our previous experiments with LePrEF and our GP variation, leading to a direct comparison with previous results obtained in this very thesis. Due to its availability, we also evaluated its impact in the whole Gov2 collection.

In order to use phrasal term information in the Learning to Rank process, we deployed two strategies: (1) We treated the phrasal terms in the same way that normal terms are treated, and their frequencies are added by the IR methods such as BM25 and VSM in the usual way, similar to our approach of expanding documents with phrasal terms . (2) We treated the phrasal terms as new features for the evidence fusion methods, giving them the chance to evaluate phrasal term information on its own as evidence of relevance. We present more information about the details of our implementation of these two alternatives in Section 5.1.

We thus evaluated the impact of phrasal terms when using Genetic Programming as the fusion method. We chose GP for this task because, not only it was the best performing method in the LETOR dataset in our experiments in Section 3.4, but it is also the basis for LePrEF.

We also evaluated the performance of LePrEF in this datasets with added phrasal term information. We believe that since LePrEF is based on a term by term approach, and thus completely disregards any co-occurrence information, it would yield a great advantage from the addition of such information.

5.1 Metodology

As previously stated, in order to assess whether phrasal terms can enhance result quality in Web search tasks with evidence fusion, we chose to use two datasets: the original LETOR benchmark dataset, which is based on Gov2, and a dataset based on previous TREC evaluations also made with Gov2, which will be referred in this thesis as **Gov2** dataset.

As in the experiments presented in Section 3.4.1, we used the LETOR MQ2007 subset, that has an average of 3.84 words per query, and about 41 documents evaluated per query.

We also performed experiments in our Gov2 dataset, which was based on previous ad hoc evaluations from TREC (Metzler et al., 2004). We used topics 700-850 as the evaluation queries. While the number of queries is less than 10% of the number present in LETOR, on the other hand each query has an average of 900 evaluated results, which can probably give a better insight on how the proposed modifications can lead not only in an enhanced top result ranking, but also its capabilities to enhance recall by bringing new results that would not be considered top notch otherwise. We split those 150 and performed a 10-fold cross validation.

From Gov2, which is the basis of both datasets, we randomly extracted 500 samples of phrasal and 500 of non-phrasal terms from the 24,824,998 bigrams present, already discounting bigrams with stop words and containing numbers, as done in Chapter 4, and used our phrasal terms detection technique to automatically detect 402,091 of them as phrasal terms. In the 1692 queries present in MQ2007, 721 had detected phrasal terms, for a total of 945 phrasal terms, of which 759 were distinct (some queries had more than one phrasal term). In Gov2's 150 queries, of the 312 possible bigrams in them, only 206 have our phrasal term restrictions (non stop word, not having numbers, at least appearing 10 times in the dataset). Of these, 92 were detected as phrasal terms by our method.

As previously stated, we deployed two approaches to adding phrasal term information to LETOR and Gov2:

a) **Updated Features** Adding their frequencies and other statistics as new terms. In this approach, the newly detected phrasal terms are considered as normal, additional terms, and their frequencies, VSM, BM25 and language model values are added to the respective textual features of LETOR. In a similar fashion, inverse document frequency and other statistics that are presented in LETOR are also updated.

b) **Extra Features** As new features pertaining phrasal terms statistics. In this approach, 30 new features derived from phrasal terms are added to LETOR: we calculated, for each of the six textual features present in LETOR (text, anchor text, title, url tokenization, title, whole document) the following features pertaining the presence of phrasal terms in documents: text frequency, inverse document frequency, VSM score, BM25 score and Language Model score.

To add this information, we had to find, for each document present in LETOR or in Gov2, its corresponding phrasal term information in order to add it to the dataset by either method. This was done by parsing these documents in GOV2 and extracting the phrasal term information pertaining only the detected phrasal terms that are present in the queries used in LETOR or Gov2. We disregarded all other phrasal terms that are present in the queries, since we showed in Table 4.11 that the impact of phrasal terms not present in the query is negligible. We found that, of the 65323 documents present in LETOR, 24274 had phrasal terms that were also in at least one of LETOR queries.

With these two approaches, we cover the most simple and direct methods to include phrasal terms in a Learn to Rank dataset. We do not dismiss, nonetheless, the possibility of more sophisticated approaches leading to better quality results. We used GP to do the evaluation, in a similar setup as presented in Section 3.2, and compared the results obtained by GP in the dataset without phrasal phrasal terms with our two approaches to add phrasal terms to this dataset.

LePrEF

We also evaluated whether the use of phrasal term information could lead to an increase in the quality of results obtained by LePrEF or not. To do so, we deployed a direct approach: Since LePrEF works on term-by-term basis, we simply included the phrasal terms information in the dataset as new terms in the same queries.

5.2 Experiments

Table 5.1 presents the results obtained by the addition of phrasal terms to LETOR. As can be seen, the addition of phrasal terms did not led to any form of quality gain. Not only that, in both NDCG

Table 5.1: Results obtained by adding phrasal terms information to LETOR

		LETOR	Updated Features	Extra Features
GP	NDCG	0.504	0.5001	0.499
	MAP	0.469	0.4661	0.4654
LePrEF	NDCG	0.491	0.490	0.488
	MAP	0.455	0.451	0.449

and MAP, it led to slight losses in quality, even though small. It is also important to notice that, while obtaining smaller NDCG and MAP values than the original dataset, updating the features to include phrasal term information instead of creating new features slightly improved the results, albeit in a non-statistically significant fashion and still underperforming in comparison to not adding any phrasal term information.

At first sight, it might seem unusual that the addition of information that has been shown in Chapter 4 to yield good results in text only search has not yielded similar gains in a scenario with multiple sources of relevance evidence. One factor that could be the cause of such behavior is that LETOR is a data collection where there is a large number of queries with very few evaluated results for each query (Liu et al., 2007) (at most 40). Thus, depending on how these documents were selected, possible gains obtained by adding phrasal terms could be diminished if potential good results that the phrasal terms would bring to the top are not present in the dataset. In order to verify whether that scenario is a real possibility or not, we indexed the Gov2 dataset (which is the base for both LETOR and the Gov2 dataset), and measured the percentage of the top results obtained by BM25 that are not present in those datasets.

Figure 5.1 shows the percentage of documents within the top 1,2,3,5,10 and 20 BM25 results in the Gov2 dataset with the addition of phrasal terms that are not present in the same query in the LETOR dataset. As can be seen, a very large number of results is missing. In the same table, we also present the same statistic regarding Gov2 dataset, and as it can be seen, the percentage is

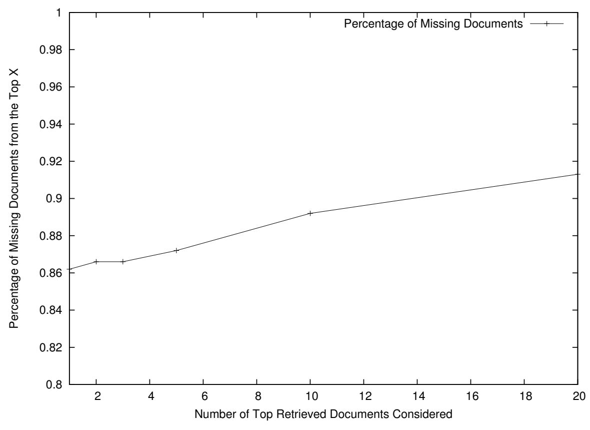

Figure 5.1: Percentage of Missing Documents in the Top Retrieved Documents from the 1692 LETOR queries

much smaller.

These results represent a strong indication that, while LETOR is a good measure of how different learning to rank methods fare in comparison to each other, LETOR in fact is unsuitable to simply adding new information to the same documents, due to its small number of evaluated results per query. In fact, LETOR was not created with the intent to be a reference collection to measure the impact of the addition of new sources of relevance evidence, being developed with the only intent to compare learning to rank methods.

In order to better assess the possible quality gains obtained when adding phrasal terms information to evidence fusion, we also performed experiments with the whole Gov2 dataset, using a different subset of previously evaluated queries. The results obtained in this dataset are presented in Table 5.2.

Table 5.2: Results obtained by adding phrasal term information to the Gov2 Collection

		Original GOV2	Updated Features	Extra Features
GP	NDCG	0.503	0.508	0.499
	MAP	0.344	0.348	0.345
LePrEF	NDCG	0.519	0.526	0.522
	MAP	0.361	0.370	0.367

Table 5.2 presents the results obtained when applying GP to the whole Gov2 collection. As expected from the results presented in Table 5.1, the fact that each query has a much larger number of evaluated results might have led to this perceived increase in quality in the index with phrasal terms. These results by themselves are already an indication that phrasal terms can indeed enhance search results, especially when updating features instead of creating new ones. This difference is consistent with what was saw in LETOR (Table 5.1), and is an indication that using this new information as an independent feature is not a good alternative.

Table 5.2 presents the results obtained by LePrEF in the Gov2 collection. The first detail that comes to attention is that, while in LETOR GP outperformed LePrEF, in the whole Gov2 collection LePrEF obtained visibly better results in terms of NDCG, with the difference between GP and LePrEF being statistically significant (p-value $<$ 0.001). Also, with the addition of phrasal terms, these results were further improved (from 0.503 to 0.509 in GP versus from 5.19 to 5.26 in LePrEF), however, this improvement may not be considered statistically significant (0.08 $<$ p-value $<$ 0.15 for all comparisons). These results are not only an indication that the inclusion of phrasal terms information can lead to better search quality, but also that LePrEF has room for improvement with the addition of co-occurrence information.

Chapter 6

Conclusions

In this thesis we propose and study a new paradigm for adopting machine learning techniques to improve quality of ranking results on search systems. While previous approaches are usually focused on applying machine learning techniques and their results directly at query processing times, such as using learning to rank techniques, we here are focused on applying machine learning **during indexing time**, essentially performing a learning to index task. The main advantage of this is that, since most of the processing is done prior to query processing, the use of such techniques may lead to many advantages while not imposing a large bump in computational costs (in fact, one of our methods aims at reducing computational costs).

To present the possible gains obtained by this approach, we proposed two machine learning techniques that are applied in the indexing stage of search engines: One aiming at reducing query processing times by performing source of evidence fusion at indexing time, effectively minimizing the effort needed to process a query, and other aimed at improving the quality of the final answer provided by such systems by means of detecting special bigrams, the phrasal terms, that have a specific semantic value when co-occurring, and adding information about those terms in the index of search engines in order to produce better quality in query processing results.

Our first approach to enhance search engine dataset indexing is our new learning to index strategy named as LePrEF, a method to fuse various sources of evidence at indexing time using genetic programming, which is a well known machine learning technique. The proposal reduces the query processing computational costs with little loss in the quality of the obtained results. Its main idea is that several sources of evidence can be fused into a single value at indexing time, leading to various simplifications in query processing. In the experiments presented using the LETOR benchmark, the loss was from 0.500 to 0.488 when compared to the best baseline found in this benchmark in terms of mean NDCG. In terms of MAP, the results varied from 0.467 to 0.452.

Besides LePrEF, we also proposed a modification to the Genetic Programming best candidate selection phase, and our modified GP obtained results that were better than those presented by state-of-the-art methods in the LETOR benchmark.

The second approach we proposed was a technique to automatically detect phrasal terms in search engine datasets, and how to include this new information in search engine indexes.

Since most popular text search models are based on a bag-of-words approach, and that is also a characteristic of LePrEF, we investigated techniques to add word relationships in IR models, in order to possibly achieve higher precision levels. We presented a machine learning method to detect phrasal terms in document collections, based solely in occurrence and co-occurrence statistics of the terms in large bodies of text. Experiments showed that the proposed method yields good results, without a large computational overhead. The classification of phrasal terms achieved values of F1 ranging from 70,80% up to 94%.

We also presented the impact that the use of the phrasal term information has on document searching, by adding this information to search engine indexes. When compared to the traditional Vector Space Model, the gain obtained by using phrasal terms was up to 11% in MAP, when considering all the queries, and up to 36% when considering only queries that do have phrasal terms. Thus, the use of phrasal terms is a strong solution to enhance the quality of search tasks while not adding a large computational overhead, since most of the processing is done at indexing time.

We also performed experiments evaluating the behavior of a search engine when both of the machine learning techniques proposed in this thesis are applied to its dataset. These results showed that the addition of LePrEF and phrasal terms in the index lead to a visibly superior index quality wise, with NDCG values of 0.526 to the original index's 0.503.

This research opens several questions that can be addressed in future work both in the direction of further improving the efficiency of the use of LePrEF and also in the direction of reducing even more the loss in the quality of results. Future directions can also be related to the usage of other machine learning methods besides genetic programming in the task of learning to index.

For instance, when considering efficiency issues the usage of LePrEF opens opportunities to new research, such as deriving pruning methods. The pruning in UTI indexes is likely to achieve higher reduction in processing costs than the ones obtained in separate indexes, since it contains global information about the final ranking, while this information is not available when the indexes are separated. This possibility should be further investigated in the future.

Another alternative to reduce the possible loss in quality, while taking advantage of the usage of UTI indexes is to adopt the LePrEF method to obtain the potentially relevant documents at the first phase of query processing, using a traditional learning method in the second phase. This alternative can be interesting, since LePrEF is more accurate in the selection of candidates, allowing a selection of a smaller, and probably better, set of candidate documents in the first phase. However, a detailed study should be performed to better assert the improvements obtained with this idea.

Most importantly, this thesis showed that applying machine learning to the indexing stages of search engines is indeed a very promising research direction, and we believe that there is still much work that can be done to improve search engines by doing new research in this direction.

Bibliography

Al-Maskari, A. and Sanderson, M. (2010). A review of factors influencing user satisfaction in information retrieval. *J. Am. Soc. Inf. Sci. Technol.*, 61(5):859–868.

Anh, V. N. and Moffat, A. (2002). Impact transformation: effective and efficient web retrieval. In *ACM SIGIR'02*, SIGIR '02, pages 3–10, New York, NY, USA. ACM.

Anh, V. N. and Moffat, A. (2005). Simplified similarity scoring using term ranks. In *ACM SIGIR'05*, pages 226–233, New York, NY, USA. ACM.

Anh, V. N. and Moffat, A. (2006). Pruned query evaluation using pre-computed impacts. In *ACM SIGIR'06*, pages 372–379, New York, NY, USA. ACM.

Anh, V. N., Wan, R., and Moffat, A. (2008). Term impacts as normalized term frequencies for bm25 similarity scoring. In *SPIRE*, pages 51–62.

Baeza-Yates, R., Calderon-Benavides, L., and Gonzalez-Caro, C. (2006). The intention behind web queries. In *Proceedings of the International Conference on String Processing and Information Retrieval*, pages 98–109, Glasgow, UK.

Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern Information Retrieval*. Addison-Wesley Professional, Boston, MA, USA.

Bendersky, M., Metzler, D., and Croft, W. B. (2010). Learning concept importance using a weighted dependence model. In *ACM WSDM'10*, pages 31–40, New York, NY, USA. ACM.

Blanco, R., Bortnikov, E., Junqueira, F., Lempel, R., Telloli, L., and Zaragoza, H. (2010). Caching search engine results over incremental indices. In *ACM SIGIR'10*, SIGIR '10, pages 82–89, New York, NY, USA. ACM.

Calado, P., Ribeiro-Neto, B., Ziviani, N., Moura, E., and Silva, I. (2003). Local versus global link information in the web. *ACM Trans. Inf. Syst.*, 21:42–63.

Cambazoglu, B. B., Zaragoza, H., Chapelle, O., Chen, J., Liao, C., Zheng, Z., and Degenhardt, J. (2010). Early exit optimizations for additive machine learned ranking systems. In *ACM WSDM'10*, pages 411–420, New York, NY, USA. ACM.

Chang, C.-C. and Lin, C.-J. (2001). *LIBSVM: a library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

da Costa Carvalho, A. L., Rossi, C., de Moura, E. S., da Silva, A. S., and Fernandes, D. (2012). Lepref: Learn to precompute evidence fusion for efficient query evaluation. *Journal of the American Society for Information Science and Technology*, 63(7):1383–1397.

de Almeida, H. M., Gonçalves, M. A., Cristo, M., and Calado, P. (2007). A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *Proc . of SIGIR '07*, pages 399–406, New York, NY, USA. ACM.

de Moura, E. S., dos Santos, C. F., santos de Araujo, B. D., da Silva, A. S., Calado, P., and Nascimento, M. A. (2008). Locality-based pruning methods for web search. *ACM Trans. Inf. Syst.*, 26(2):9:1–9:28.

Ekkerman, R. B. and Allan, J. (2003). Using bigrams in text categorization. In *Tech Report*, pages 1–10.

Elias, P. (1975). Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(1):194–203.

Fan, W., Fox, E. A., Pathak, P., and Wu, H. (2004a). The effects of fitness functions on genetic programming-based ranking discovery for web search. *JASIST*, 55(7):628–636.

Fan, W., Gordon, M., Pathak, P., Xi, W., and Fox, E. (2004b). Ranking function optimization for effective web search by genetic programming: An empirical study. In *System Sciences, Proc. of the 37th Hawaii International Conference on System Sciences (HICSS'04)*, volume 4. IEE CNF.

Fan, W., Gordon, M. D., and Pathak, P. (2004c). Discovery of context-specific ranking functions for effective information retrieval using genetic programming. *IEEE Transactions on Knowledge and Data Engineering*, 16(4):523–527.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969.

Hersh, W., Buckley, C., Leone, T. J., and Hickam, D. (1994). Ohsumed: an interactive retrieval evaluation and new large test collection for research. In *Proceedings of the International ACM SIGIR Conference on Research & Development of Information Retrieval*, pages 192–201, New York, NY, USA. Springer-Verlag New York, Inc.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *ACM SIGKDD'02*, pages 133–142, New York, NY, USA. ACM.

Liu, T.-Y., Xu, J., Qin, T., Xiong, W., and Li, H. (2007). LETOR: Benchmark Dataset for Research on Learning to Rank for Information Retrieval. In *LR4IR 2007, in conjunction with SIGIR 2007.*

Metzler, D., Strohman, T., Turtle, H., and Croft, W. (2004). Indri at trec 2004: Terabyte track. Technical report, DTIC Document.

Miller, G. A., Beckwith, R., Fellbaum, C., Gross, D., and Miller, K. (1990). Wordnet: An on-line lexical database. *International Journal of Lexicography*, 3:235–244.

Mladenic, D. and Grobelnik, M. (1998). Word sequences as features in text-learning. In *In Proceedings of the 17th Electrotechnical and Computer Science Conference (ERK98*, pages 145–148.

Mosberger, D. and Jin, T. (1998). httperf—a tool for measuring web server performance. *SIGMETRICS Perform. Eval. Rev.*, 26:31–37.

Persin, M., Zobel, J., and Sacks-Davis, R. (1996). Filtered document retrieval with frequency-sorted indexes. *JASIST*, 47:749–764.

Pôssas, B., Ziviani, N., Meira, Jr., W., and Ribeiro-Neto, B. (2005). Set-based vector model: An efficient approach for correlation-based ranking. *ACM Trans. Inf. Syst.*, 23(4):397–429.

Saraiva, P. C., Silva de Moura, E., Ziviani, N., Meira, W., Fonseca, R., and Riberio-Neto, B. (2001). Rank-preserving two-level caching for scalable search engines. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '01, pages 51–58, New York, NY, USA. ACM.

Silva, T. P. C., de Moura, E. S., Cavalcanti, J. M. B., da Silva, A. S., de Carvalho, M. G., and Gonçalves, M. A. (2009). An evolutionary approach for combining different sources of evidence in search engines. *Inf. Syst.*, 34:276–289.

Soboroff, I. (2002). Do trec web collections look like the web? *SIGIR Forum*, 36(2):23–31.

Tan, C.-M., Wang, Y.-F., and Lee, C.-D. (2002). The use of bigrams to enhance text categorization. *Inf. Process. Manage.*, 38(4):529–546.

tau Yih, W., Goodman, J., and Carvalho, V. R. (2006). Finding advertising keywords on web pages. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 213–222, New York, NY, USA. ACM.

Tesar, R., Strnad, V., Jezek, K., and Poesio, M. (2006). Extending the single words-based document model: a comparison of bigrams and 2-itemsets. In *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering*, pages 138–146, New York, NY, USA. ACM.

Trotman, A. (2005). Learning to rank. *Information Retrieval*, 8(3):359–381.

Voorhees, E. M. (1999). Overview of the eighth text retrieval conference. In *Proceedings of the Text REtrieval Conference*, pages 1–24, Gaithersburg, Maryland.

Wang, C., Bi, K., Hu, Y., Li, H., and Cao, G. (2012). Extracting search-focused key n-grams for relevance ranking in web search. In *Proceedings of the fifth ACM international conference on Web search and data mining*, WSDM '12, pages 343–352, New York, NY, USA. ACM.

Wang, L., Lin, J., and Metzler, D. (2010). Learning to efficiently rank. In *ACM SIGIR'10*, pages 138–145, New York, NY, USA. ACM.

Witten, I. H. and Frank, E. (2000). *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufman.

Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. (2008). Listwise approach to learning to rank: theory and algorithm. In *ICML'08*, pages 1192–1199, New York, NY, USA. ACM.

Xu, J. and Li, H. (2007). Adarank: a boosting algorithm for information retrieval. In *ACM SIGIR'07*, pages 391–398, New York, NY, USA. ACM.

yuan Yeh, J., yi Lin, J., ren Ke, H., and pang Yang, W. (2007). Learning to rank for information retrieval using genetic programming.

Zhang, W., Liu, S., Yu, C., Sun, C., Liu, F., and Meng, W. (2007). Recognition and classification of noun phrases in queries for effective retrieval. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 711–720, New York, NY, USA. ACM.

Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM Comput. Surv.*, 38.

Zongker, D. and Punch, B. (2006). lilgp 1.01 user's manual. Technical report, Michigan State University.