APRENDIZADO ATIVO PARA ORDENAÇÃO DE RESULTADOS

RODRIGO DE MAGALHÃES SILVA

APRENDIZADO ATIVO PARA ORDENAÇÃO DE RESULTADOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Orientador: Marcos André Gonçalves
Coorientador: Adriano Veloso

Belo Horizonte
Maio de 2012

RODRIGO DE MAGALHÃES SILVA

ACTIVE LEARNING FOR LEARNING TO RANK

Dissertation presented to the Graduate Program in Computer Science of the Universidade Federal de Minas Gerais - Departamento de Ciência da Computação in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Marcos André Gonçalves
Coadvisor: Adriano Veloso

Belo Horizonte
May 2012

Silva, Rodrigo de Magalhães

S586a Active Learning for Learning to Rank / Rodrigo de Magalhães Silva. — Belo Horizonte, 2012
xxii, 104 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais - Departamento de Ciência da Computação
Orientador: Marcos André Gonçalves
Coorientador: Adriano Veloso

1. Computação - Teses. 2. Recuperação de Informação - Teses. I. Orientador. II. Coorientador. III. Título.

CDU 519.6*73(043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Aprendizado ativo para ordenação de resultados

RODRIGO DE MAGALHÃES SILVA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

[illegible]
PROF. MARCOS ANDRÉ GONÇALVES - Orientador
Departamento de Ciência da Computação - UFMG

[illegible]
PROF. ADRIANO ALONSO VELOSO
Departamento de Ciência da Computação - UFMG

[illegible]
PROFA. GISELE LOBO PAPPA
Departamento de Ciência da Computação - UFMG

[illegible]
PROF. NIVIO ZIVIANI
Departamento de Ciência da Computação - UFMG

[illegible]
PROF. PÁVEL PEREIRA CALADO
Instituto Superior Técnico - Universidade Técnica de Lisboa

Belo Horizonte, 31 de maio de 2012.

Abstract

Ranking is an essential feature of many applications: from web search to product recommendation systems and online advertising, results have to be ordered based on their estimated relevance with respect to a query or based on a user profile or personal preferences. Ranking is especially important in Web Information Retrieval where a simple query can return hundreds of thousands or millions of documents, but the user will probably only scan the first few results. Learning to Rank (L2R) algorithms use Machine Learning (ML) techniques to provide better rankings than those obtained by classic Information Retrieval models, such as BM25. L2R algorithms need a labeled training set to generate a ranking model that can be later used to rank new query results. These training sets are very costly and laborious to produce, requiring human annotators to assess the relevance or order of the documents in relation to a query. Active learning algorithms are able to reduce the labeling effort by selectively sampling an unlabeled set and choosing data instances that maximize a learning function's effectiveness. In this work we propose a novel associative rule-based active learning method that can be used to actively select document instances from an unlabeled set without the need of having an initial seed training set. Used alone, this method provides very small and yet effective training sets. Although effective, this method is not easily extended to select more instances if necessary. Thus, another contribution of this work is a round-based second stage selection method can be used to select more instances for labeling as needed. This second stage method initially uses the small sets selected by the first method to train a committee of distinct L2R algorithms. It then progresses iteratively in rounds selecting, for each query, those documents that the committee members most disagree about in ranking. Together, these complementary methods are very powerful and flexible making the resulting combined method applicable to many real-world scenarios. We test our methods with various benchmarking datasets and compare them to several baselines to show that they achieve very competitive results using only a small portion of the original training sets. For instance, in half of the datasets tested, while selecting less than 5% of the original training sets, the proposed combined method achieves results that are better than state-of-the-art L2R algorithms trained on the complete training sets and that also surpass a strong active learning for L2R baseline method.

Resumo

A ordenação de resultados é uma funcionalidade essencial de muitos sistemas: desde máquinas de busca na Web, passando por sistemas de recomendação de produtos e propaganda virtual, resultados devem ser ordenados de acordo com sua relevância para a busca efetuada ou baseado em preferências do usuário ou em seu perfil. Essa ordenação é especialmente importante para sistemas de busca na Web, uma vez que uma simples pesquisa pode retornar centenas de milhares ou milhões de documentos, mas o usuário provavelmente irá visualizar apenas alguns poucos no topo da lista. Algoritmos que aprendem a ordenar (Learning to Rank - L2R) usam técnicas de aprendizado de máquina para produzir ordenações melhores do que aquelas obtidas por modelos clássicos de Recuperação de Informação, como o BM25, por exemplo. Métodos L2R necessitam de um conjunto de treino rotulado para poderem gerar um modelo de ordenação que pode depois ser utilizado para ordenar os resultados de novas buscas. Esses conjuntos de treino são difíceis e caros de produzir, pois é necessário que especialistas humanos avaliem os documentos retornados, indicando se são ou não relevantes ou então provendo uma ordem parcial ou total desses documentos de acordo com sua relevância para a busca. Algoritmos de aprendizado ativo podem reduzir o custo desse processo de análise de relevância, selecionando de um conjunto não-rotulado de instâncias que têm o potencial de acelerar o aprendizado do algoritmo de L2R, maximizando a eficácia do modelo aprendido. Nessa dissertação, propomos um novo método de aprendizado ativo baseado em regras de associação que pode ser usado para selecionar documentos de um conjunto não-rotulado sem a necessidade de um conjunto inicial de treino. Esse método seleciona conjuntos de treino pequenos e efetivos. Entretanto, não é simples estender esse método para selecionar mais documentos caso isso seja necessário. Portanto, outra contribuição desse trabalho é um método iterativo de seleção de documentos que pode ser usado para selecionar mais instâncias para serem rotuladas. Nesse segundo estágio, inicialmente usamos os documentos selecionados pelo primeiro método para treinar um comitê de algoritmos L2R distintos. Depois, o método progride iterativamente selecionando, para cada busca, aqueles documentos cuja ordem mais varia entre as ordens propostas pelos membros do comitê. Usados em conjunto, esses dois métodos são poderosos e flexíveis, podendo ser aplicados em vários cenários reais. Testamos os métodos propostos utilizando vários conjuntos de teste da coleção LETOR e mostramos que eles obtém resultados competitivos selecionando uma pequena fração dos conjuntos de treino originais. Em metade dos conjuntos de teste avaliados, selecionando menos de 5% dos treinos originais, a combinação dos métodos obtém resultados melhores do que aqueles obtidos por algoritmos L2R do estado-da-arte utilizando os treinos completos. Também mostramos que a combinação dos métodos é melhor do que um algoritmo de aprendizado ativo para L2R recentemente proposto na literatura.

List of Figures

2.1 Ranking Framework 8

2.2 Learning to Rank Framework 17

2.3 SVMRank: two-dimensional example of how two possible weight vectors order four documents 27

3.1 Active Learning Framework 33

4.1 % Selected samples vs. # of Bins (left) and MAP vs. # of Bins (right) . . 52

4.2 % of instances selected per partition as we vary the number of partitions . 54

4.3 % Selected samples vs. # of Partitions (left) and MAP vs. # of Partitions (right) 55

4.4 Comparison of ARLR, SVMS and Donmez: MAP vs. % of samples selected from unlabeled set for 25 rounds 60

5.1 TD2003 and TD2004: ARLR-QBC and baselines comparison. MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set . . . 69

5.2 HP2003, HP2004, NP2003 and NP2004: ARLR-QBC and baselines comparison. MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set 71

5.3 TD2003 and TD2004: Comparing RR, RReal and ARLR initial selection strategies; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set 74

5.4 HP2003, HP2004, NP2003 and NP2004: Comparing RR, RReal variations and ARLR initial selection strategies; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set 76

5.5 TD2003 and TD2004: Comparing RR, RReal, RRealOracle and ARLR initial selection strategies with QBC; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set 77

5.6 HP2003, HP2004, NP2003 and NP2004: Comparing RR, RReal, RRealOracle and ARLR initial selection strategies with QBC; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set 79
5.7 All datasets: Proportion of relevant instances in the selected sets (left) and the percentage they represent of the total number of relevant samples in the unlabeled sets (right) 81
5.8 TD2003 and TD2004: Ranking using RLR with the selected sets; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set 84
5.9 HP2003, HP2004, NP2003 and NP2004: Ranking using RLR with the selected sets; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set 85
5.10 Comparison of ARLR-QBC, ARLR and three LETOR baselines: Best MAP obtained in rounds 0 to 7 88

List of Tables

2.1 LETOR 3.0 Web Datasets' Characteristics 20

4.1 ARLR without Partitioning MAP Results and Statistics 46
4.2 ARLR with 5 Partitions MAP Results and Statistics 50
4.3 MAP for RLR using the complete sets with different discretizations 53
4.4 MAP for SVM using selected samples, BM25 and Random Baselines . . . 57
4.5 MAP for ARLR and LETOR Baselines 58
4.6 NDCG for ARLR and LETOR Baselines: ARLR vs. Rankboost and FRank (a), ARLR vs. Regression and SVMRank (b) 59

5.1 Gains obtained by ARLR-QBC over RReal-Donmez and SVMRank: MAP (a) and NDCG@10 (b). Average Gain for rounds 0-7 (AG7%), Max Gain for rounds 0-7 (MG7%), Average Gain for all rounds (AG%) and Max Gain for all rounds (MG%). The number in parentheses indicate in which round the maximum gain was obtained. Values in *italic* indicate negative gains, values in **bold** indicate a gain over 5% and values in ***italic and bold*** a gain over 10% 89

List of Algorithms

1 Rule-based Learning to Rank (RLR) 24
2 Active Rule-based Learning to Rank: ARLR 45
3 ARLR with Partitioning 49

Contents

Abstract **ix**

Resumo **xi**

List of Figures **xiii**

List of Tables **xv**

List of Algorithms **xvii**

1 Introduction **1**

1 Context and Motivation 2

2 Contributions 3

3 Document Organization 4

2 Ranking and Learning to Rank **7**

1 Ranking 7

1.1 Introduction 7

1.2 Classic Ranking Models for IR 8

2 Measuring Ranking Quality 11

2.1 Evaluation Framework 12

2.2 Common Metrics 13

2.3 Benchmarking Datasets 15

3 Learning to Rank 16

4 Benchmarking Datasets for L2R 19

5 Learning to Rank Using Association Rules: RLR 21

5.1 Rule-based Learning to Rank 21

5.2 Demand-Driven Rule Extraction 22

5.3 Relevance Estimation 23

6 Other Supervised Learning to Rank Algorithms 24
6.1 SVMRank 25
6.2 RankBoost 27
7 Summary 29

3 Active Learning 31
1 The Labeling Effort 31
2 Reducing the Labeling Effort: Active Learning 32
3 Active Learning Strategies 34
4 Related Work 35
4.1 Active Learning for Learning to Rank 35
4.2 The Donmez Method 37
5 Summary 40

4 Active Learning to Rank Using Association Rules 41
1 Introduction 41
2 Active Learning using Association Rules: ARLR 42
3 Experimental Evaluation 45
3.1 Experimental Setup 45
3.2 Results 46
3.3 Analysis 50
4 Summary 60

5 Expanding Selection with Query-By-Committee 63
1 Introduction 63
2 Stage 1: Active Learning using Association Rules 64
3 Stage 2: Query-By-Committee 65
3.1 QBC Active Learning 65
3.2 Measuring Rank Disagreement 66
4 Experimental Evaluation 67
4.1 Experimental Setup 67
4.2 Baselines 68
4.3 Results 69
4.4 Analysis 73
4.5 Using RLR with the selected sets 84
4.6 Comparing ARLR-QBC to LETOR baselines 87
4.7 Gains obtained over RReal-Donmez and SVMRank 88
5 Summary 90

6 Conclusions **93**
1 Future Work 93

Bibliography **97**

Chapter 1

Introduction

We live in The Age of Information. The advent and impressive growth of the Web has brought about a deluge of information that is increasingly accessible through networked devices anytime, anywhere. Individuals, companies, government and other institutions are publishing and consuming an enormous amount of information on a daily basis. The distributed nature of this publishing process and the ease-of-use of its underlying technologies and protocols has allowed the Web to develop and grow as fast as it does but has also made harder the problem of efficiently and effectively searching and accessing relevant information. The development and adoption of the Web as an information sharing platform has spurred new research and technology to help dealing with the challenge of sieving through the data to quench our information needs. As a consequence, the field of Information Retrieval has seen major new developments in the last two decades.

Information Retrieval, or IR, concerns the technologies and methods for retrieving from a collection of information items those that satisfy an *information need* expressed by a user. A Web search engine, for instance, uses IR techniques to index and make available for search a portion of the information items accessible through the Web. These information items are usually web pages containing text, but they can also be images, videos or other structured or semi-structured data. The plain size of the Web poses enormous practical challenges on the design, implementation and operation of a web search system. Not only the size of such "collection" is of the order of tens of billions of documents, but a good part of its content is constantly changing: new content is being added and old content being updated or removed.

An essential function of an IR system is to be able to *rank* the results returned for an user query in such a way as to maximize the chance that the most relevant documents for the query appear at the top of the ranked list. In fact, the concept of *ranking* is at the core of some of the classic IR methods that we will briefly describe in Chapter 2. In the last ten years or so, there has been growing interest in applying Machine Learning (ML) techniques to the ranking problem. ML techniques have been successfully applied to many difficult problems in the past. We are particularly interested in *supervised learning* where past evaluations of the relevance of documents ranked by some IR system (what we call *training data*) is used to train a ranking system so that the resulting *learned model* or function can be used to better rank documents returned by new queries.

These Learning to Rank (L2R) methods have shown to be very effective, providing better rankings than the classic IR approaches. But in order to be able to use a L2R method one needs to have *training data*: query-document data instances that are annotated by human assessors indicating whether they are relevant to the query. In supervised learning, these are known as the *labeled examples* from which the learning algorithm can derive a function or model that can be later used to rank documents returned as result to a new query.

1 Context and Motivation

Learning to Rank algorithms rely on labeled or ordered training sets to build ranking models that are used to rank results at query time. To create these training sets, human annotators must somehow evaluate the documents returned by a set of queries. This can be done by experts that analyze each document in turn, or by users of an IR system through some sort of Relevance Feedback (RF) mechanism such as clickthrough data (see, for instance, Joachims [2002] and Radlinski and Joachims [2007]). Depending on what type of rank learning algorithm is used, it may be necessary to provide a binary relevance judgment (i.e. relevant, not relevant), a relevance level (e.g., somewhat relevant, very relevant, extremely relevant), a pairwise ordering (e.g. document d_i is more relevant than document d_j or $d_i \succ d_j$) or a complete or partial ordering of the documents returned by a query (e.g. $d_i \succ d_j \succ [...] \succ d_k$). These different relevance judgment types correlate to the three main approaches used by L2R methods; namely pointwise, pairwise and listwise.

Quality training data is necessary for learning to rank algorithms to be effective. Yet, it is expensive to produce any amount of it. Active learning techniques have been proposed to help dealing with the labeling effort problem. The motivation behind active learning is that it may be possible to achieve highly effective learning functions by carefully selecting and labeling instances that are “informative” to the learning algorithm. Using active learning, we can reduce the cost of producing training sets for rank learning algorithms and even improve the effectiveness of the learned functions by avoiding adding “noisy” instances to the training sets. Furthermore, human annotators can spend more time analyzing the relevance of each selected instance, producing better training data.

In a typical active learning scenario, data instances are selected from an unlabeled set one at a time and labeled by a human expert. Every time a new sample is selected and labeled, a new learning model is produced and the active learning algorithm again chooses a new instance from the unlabeled set. This process can be repeated as long as necessary, until the labeling budget is exhausted or until the learned model achieves a some predefined quality or property. The product of an active learning method is a small and yet highly effective training set that can be used by supervised learning algorithms to rank new user queries.

2 Contributions

In this work we propose two new active learning techniques for L2R. The first, detailed in Chapter 4, uses on-demand associative rules to sample an unlabeled set, selecting document instances based on a simple diversity principle. This technique is highly effective, obtaining very good results and selecting extremely small training sets. Although effective, this method is not easily extended to select more instances if necessary. If, after using ARLR, there’s still labeling budget available or for some other reason it is possible or desirable to select and label more instances, an extension of the method is necessary. In Chapter 5 we propose a round-based second stage procedure based on a Query-By-Committee (QBC) process that allows for the selection and labeling of as many instances as desired or possible, given the available resources.

These two techniques complement each other. Using them, it is possible to build a learning to rank training set from scratch and use it to effectively rank results from new queries. The proposed methods are practical and easily applicable in a real-world scenario where no previous training data is available.

In summary, the main contributions of this work are:

- An Active Rule-based Learning to Rank (ARLR) method that can be used to actively select document instances from an unlabeled set without the need of having an initial seed training set. This work has been published in Silva et al. [2011]. Used alone, ARLR provides very small and yet effective training sets with the advantage that it naturally stops selecting instances. This characteristic may be useful in certain real-world situations where all that is needed is a small training set to bootstrap the learning process.

- A round-based second stage selection method (QBC) that can be used to select more instances for labeling as necessary. This second stage method initially uses the small sets selected by ARLR to train a committee of distinct L2R algorithms. It then progresses iteratively in rounds selecting, for each query, those documents that the committee members most disagree about in ranking. Together, these complementary methods are very powerful and flexible making the resulting combined method applicable to many real-world scenarios.

3 Document Organization

Before discussing the novel active learning techniques presented in this work, we briefly introduce some key concepts related to Ranking, Learning to Rank and Active Learning. This is done in order to familiarize the reader with some of the concepts that will later be relied upon when presenting and analyzing the proposed active learning methods. The reader may choose to skip some of the material in the second and third chapters.

Chapter 2 starts with a general description of the ranking problem and a quick primer on the classic ranking models. We then lay out a general framework for evaluating a ranking algorithm and explain some of the most common metrics used to evaluate ranking quality. We also discuss Learning to Rank, detailing a simplified L2R framework and glossing over the three main categories of learning to rank methods. We then describe the LEarning TO Rank (LETOR) benchmarking datasets that we use in the experimental evaluation of the methods proposed in this work. This Chapter also provides an overview of the Rule-based Learning to Rank (RLR) supervised method proposed by Veloso et al. [2008], from which we derived the idea of the active learning method we call ARLR. Finally, we give a brief description of two well known supervised L2R algorithms that are used in the QBC strategy described in Chapter 5.

In Chapter 3 we start by further discussing the motivation behind active learning and then briefly describe some of the active learning strategies that have been developed by ML researchers over the years. We then present a review of active learning methods designed specifically for learning to rank. Finally, we describe in some detail one of these methods which we later use as an active learning to rank baseline.

In Chapter 4 we present a novel Active Rule-based Learning to Rank (ARLR) method. The chapter starts by explaining how we derive a diversity-based active learning strategy from RLR. We then present the results obtained by extensive experimentation of ARLR on the LETOR 3.0 datasets, and discuss some practical aspects and extensions of the method. We compare the results with several baselines (both active learning methods and supervised methods), showing that ARLR obtains very competitive results selecting extremely small training sets.

Chapter 5 proposes a two-stage hybrid active learning approach where ARLR is used to select small initial training sets that are then used by a round-based Query-By-Committee (QBC) active method to expand the selection as needed or desired. This novel method is not only very effective, but also very flexible and practical. We again describe the extensive experimentation performed using this new technique and analyze the results in depth, proposing some variations and improvements.

Finally, in Chapter 6 we summarize the main aspects of this work and trace some of the future directions that can be pursued based on what we have learned from the current work.

Chapter 2

Ranking and Learning to Rank

1 Ranking

1.1 Introduction

Ranking is an essential feature of many applications: from web search to product recommendation systems and online advertising, results have to be ordered based on their estimated relevance with respect to a query or based on a user profile or personal preferences. Ranking is especially important in Web Information Retrieval where a simple query can return hundreds of thousands or millions of results, but the user will probably only scan the first few results (according to Granka et al. [2004], usually only the first six or seven results). Thus ranking the results to a query in such a manner as to effectively put the most relevant items on the top of the list is of utmost importance.

Ranking may be the the most visible part of an Information Retrieval system (especially if it fails to actually put very relevant results in the first few positions), but it is not the only one. An IR system must first *index* the collection of documents in such a way as to make the process of retrieving the relevant documents efficient and effective. For web search engines, this task can be daunting, since it must first collect or *crawl* the Web, storing what amounts to be almost a copy of it before the indexing process can take place. The indexing process is done so as to create a *logical view* of the documents in the collection that allows for the actual process of retrieving results for a query to be efficient and effective. Figure 2.1 shows a simplified diagram of this process. The collected documents are indexed by the IR system, generating the “Logical View”. A user poses a query to the IR system, which uses the logical view to find the documents relevant to the query. The IR system can then process the documents that were retrieved (generating, for instance, text snippets to be presented

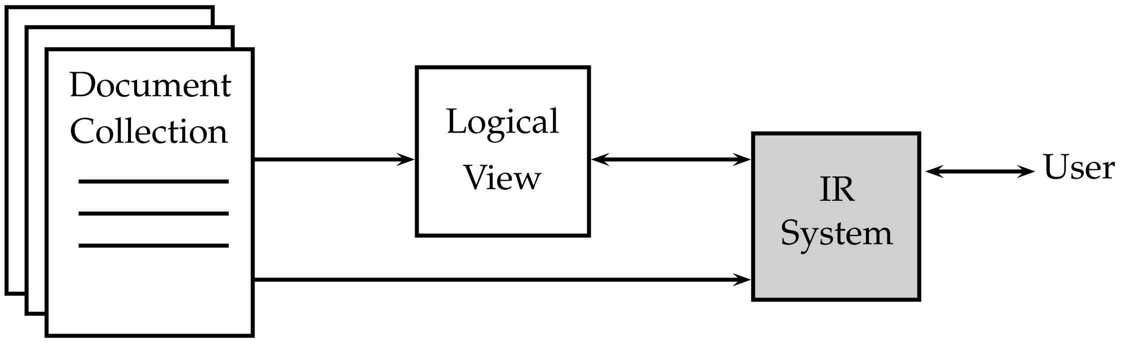

Figure 2.1. Ranking Framework

in the result interface) and gather pointers so as to be able to retrieve the actual documents that the user decides to view. There are many other processes that are not shown in the figure. In Web IR systems, the process of collecting the documents and generating their logical views is never ending: the system is constantly crawling for new pages or updating already indexed documents. The query process itself is usually much more complex, with some sort of query expansion mechanism modifying the user query to improve its retrieval efficiency. For our purposes, this simple diagram will suffice. In Section 3 we will expand the diagram showing how a Learning to Rank system relates to this simple framework.

Before that, in the next section, we briefly describe some of the most well known IR models so the reader may understand how the indexing process works and how it relates to the actual retrieval and ranking of query results. As we will see in Section 3, learning to rank systems are usually used to *re-rank* some of the results that were retrieved (and possibly ranked) by the underlying IR system. We also briefly describe an evaluation framework for ranking quality in Section 2, presenting some of the most common metrics used to evaluate the quality of ranking (and learning to rank) algorithms. Readers already familiar with the field of IR can skip the next two sections.

1.2 Classic Ranking Models for IR

Ranking is an integral part of most Information Retrieval systems. In fact, according to Baeza-Yates and Ribeiro-Neto [2011], "[t]he fundamental premises that form the basis of a ranking algorithm determine the IR model". That is, we design and implement IR systems with the ranking problem in mind. In that work, the authors identify three groups of classic IR Models, namely Boolean, Vector and Probabilistic. Let's first define some basic concepts in order to give a brief primer on the main models.

Given a collection of documents, we call $\mathcal{D}$ the set composed of logical views or representations of the documents in the collection. These logical views are usually composed of some representation of *index terms* or keywords; that is, each document is described by a set of index terms. An index term can be a word or a group of consecutive words. All index terms in the collection form the set $\mathcal{V} = \{k_1, ..., k_t\}$, of distinct terms called the *vocabulary* $\mathcal{V}$ (of size t) of the collection. These concepts allow us to represent a document d_j or a query q_i as patterns that indicate, for example, which index terms appear in that specific document or query. A pattern $d_j = (1, 0, ..., 0)$ as representation of document d_j could mean that this particular document contains the index term k_1 and no other. The same could be said of a query q_i represented by the same pattern. What this logical view of documents (or queries) gives us is the ability to represent the collection as a matrix of documents vs. index terms. The values that appear at each position in the matrix can be a simple boolean value (as in the example above) or some more informative number such as the frequency of the term in the document.

The first classic model, the Boolean Model, uses a formulation similar to the example above, where each document is represented by a pattern indicating the presence or absence of each term of the vocabulary. Queries are composed of index terms connected by the boolean operators *and*, *or* and *not*. By rewriting the query into a disjunctive form it is simple to retrieve all the documents which conform to the query. That is, the documents considered *relevant* to the query. Incidentally, the Boolean Model does not provide a ranking function; it only allows for the retrieval of all relevant documents without any measure of how relevant they are. This limits its applicability in many real-world scenarios, especially with large collections where the number of relevant documents returned by a query can be large.

Instead of only storing the information of whether a term appears or not in a document (or query), we can store some other value indicating a weight $w_{i,j}$ for each term k_i in relation to a document d_j. In this scenario, the document representation becomes $d_j = \{w_{1,j}, ..., w_{t,j}\}$ and, in the same fashion, a query is represented as $q_l = \{w_{1,l}, ..., w_{t,l}\}$. These weights can be defined in many different ways but the central concept is the same: not all terms have equal importance. A widely used term weighting scheme is known as TF-IDF (Term Frequency - Inverse Document Frequency). The term frequency (TF) measures how often a term appears in a document. The rationale behind it being that terms that appear many times in a document are important to describe the key topics of that document. A common formulation used for the TF of a term k_i appearing in a document d_j is $TF_{i,j} = 1 + \log f_{i,j}$ where $f_{i,j}$ is the number of times the term k_i appears in document d_j. If a term k_l is not present in the document

d_j, then $TF_{l,j} = 0$. The IDF, on the other hand, tries to "weight" in the fact that some terms are much more frequent than others in the collection and, thus, their informational value varies. Terms that appear in many documents are less valuable for distinguishing one document from another. A common formulation of the IDF is as follows:

$$IDF_i = \log \frac{N}{n_i} \tag{2.1}$$

Where N is the total number of documents in the collection and n_i is the number of documents containing the term k_i.

Thus, if we choose to use the TF-IDF as the weight $w_{i,j}$ associated to the pair (k_i, d_j), then one possible definition would be:

$$w_{i,j} = (1 + \log f_{i,j}) \times \log \frac{N}{n_i} \tag{2.2}$$

We say "possible definition" because there are many variations of TF, IDF and TF-IDF proposed in the literature.

Another classic IR model, the Vector Model, uses document and query representations based on weights to allow for partial matching and to provide an intrinsic ranking scheme. Using weights for document and query terms, it is possible to compute a *degree of similarity* between each document and a query. By sorting the retrieved documents using this degree of similarity, the model provides ranked lists of documents as responses to user queries. In the vector model, the index term weights of both documents and queries are represented as unit vectors of a t-dimensional space: $\vec{d_j} = (w_{1,j}, ..., w_{t,j})$ and $\vec{q} = (w_{1,q}, ..., w_{t,q})$. The degree of similarity of a document d_j in relation to a query q is defined as the correlation of the vectors $\vec{d_j}$ and $\vec{q}$. The cosine of the angle between these two vectors can be used as a measure of this correlation:

$$sim(d_j, q) = \frac{\vec{d_j} \bullet \vec{q}}{|\vec{d_j}| \times |\vec{q}|} = \frac{\sum_i^t w_{i,j} \times w_{i,q}}{\sqrt{\sum_i^t w_{i,j}^2} \times \sqrt{\sum_i^t w_{i,q}^2}} \tag{2.3}$$

Since $w_{i,j} \geq 0$ and $w_{i,q} \geq 0$, then $0 \leq sim(d_j, q) \leq 1$. Therefore, not only the vector model allows for *partial* matching between the query and a document (i.e. not all query terms must appear in a retrieved document), but it also provides a ranked list of documents, if we order the retrieved items in decreasing order of $sim(d_j, q)$. Besides using the TF-IDF as the term weights, the vector model also incorporates document length normalization (through the term $|\vec{d_j}|$), which is important since very long documents will not only have more terms, but also more instances of these terms.

The final group of classic IR models use a probabilistic ranking principle. The general idea behind this class of IR models is that the querying process specifies the properties on an ideal answer set R that should contain all documents relevant to the query and no other. The problem is that the query *per se* is not enough to provide an accurate description of this ideal answer set. The model tries to estimate the probability that, given a user query q, the user will find a document d_j interesting or relevant. The model could use an initial guess or estimate of these characteristics and then iteratively improve them using the user's feedback on the relevance of the retrieved documents. This is cumbersome and rarely done. By simplifying the requirements and assuming that initially it is not feasible to estimate the set R, the equation for ranking computation in the probabilistic model becomes

$$sim(d_j, q) \sim \sum_{k_i \in q \wedge k_i \in d_j} \log \left(\frac{N - n_i + 0.5}{n_i + 0.5} \right) \tag{2.4}$$

The above formula is known as BM1. While there is an IDF component present, neither TF weights nor document length normalization are present in the equation above. These elements were eventually added to the BM15 (TF) and BM11 (document length normalization) which were combined to form the BM25 ranking formula:

$$BM25(d_j, q) = \sum_{i=1}^{m} \frac{IDF_i \times TF_{i,j} \times (k_1 + 1)}{TF_{i,j} + k_1 \times \left(1 - b + b \times \frac{len(d_j)}{avg\_dlen}\right)} \tag{2.5}$$

where $len(d_j)$ is the length in words of document d_j, avg_dlen is the average document length in the collection and k_1 and b are free parameters that allow the formula to be tuned to a given collection. The BM25 model is widely used and, if correctly tuned, yields very good results in general collections.

2 Measuring Ranking Quality

There are many other IR models that were proposed throughout the years and that we do not discuss here. Several are variations and extensions of the basic models discussed above. When a new model or variation is developed, it is important to be able to evaluate it and analyze how it fares compared to the already established models. To this end, we need and evaluation framework that can be used to set up experiments that will allow us to compare different models. We also need measures or metrics to estimate how good the ranking produced by one method is compared to the other methods. Finally, it is useful to have benchmarking collections so that researchers can easily share results: with these collections, we can use published results as baselines to evaluate new systems and models without having to implement each model proposed in the past. In this section we describe a basic evaluation framework for ranking systems, provide definitions of some common ranking metrics and briefly discuss benchmarking datasets (we describe some L2R benchmarking datasets in Section 4).

2.1 Evaluation Framework

In order to be able to evaluate a ranking algorithm's effectiveness we need to have a well defined process; a benchmarking procedure that allows us to assess how good a ranking model is as compared to other models. This procedure could be as follows:

- Produce or collect a number of queries to form a test set. The queries can be selected randomly from a query log or chosen as representatives of types of queries.
- For each query q:
  - Retrieve a number of documents associated with the query using some retrieval technique. This could be done using a classic IR method such as the vector model or BM25. It could be done using one's own IR or ranking method or a combination of various methods.
  - Judge the relevance of each retrieved document in relation to the query. The relevance judgment needs to be performed by human assessment.
  - Use two or more ranking algorithms to rank the documents
  - Using the relevance judgment associated with each document to measure how well each ranking method performs for the current query q.
- Use an average measurement on all queries to evaluate the performance of each ranking method in comparison to the other (or others).

For the time being, let's assume that the relevance judgment can be binary (e.g. 0 = not relevant, 1 = relevant) or picked from a discrete set of possibilities (e.g. 0 = not relevant, 1 = somewhat relevant, 2 = relevant, 3 = very relevant). In section 3, we will see that there are other ways of defining the relevance, according to the type of learning to rank algorithm used.

To be able to use the described procedure, we need to have metrics to measure the ranking quality of each query and to measure the average quality of the rankings of all queries. Some of the most used ranking metrics are described below.

2.2 Common Metrics

In this section we describe some common metrics used to evaluate the quality of a ranking method. These are the metrics that we will later use to evaluate our proposed methods. They were chosen simply because they are widely used and some of the published results that we will later compare to our own have used one or more of these metrics.

Precision, Recall and P@k

Two very common metrics in IR are Precision and Recall. Precision is the proportion of the retrieved documents that are considered relevant. If the set of the relevant documents is R and the set of documents returned by the retrieval function is A, then

$$Precision = \frac{\mid R \cap A \mid}{\mid A \mid} \tag{2.6}$$

Recall is the fraction of the relevant documents that was returned by the retrieval function:

$$Recall = \frac{\mid R \cap A \mid}{\mid R \mid} \tag{2.7}$$

For web search ranking or ranking of very large collections, these metrics are hard to estimate, since we do not usually know how many documents are relevant to a query. Furthermore, is such situations, there are usually a large number of results (i.e. the set A is large), but a user is unlikely to browse through more that 10 or 20 results. Instead of calculating the overall precision, we can measure the average precision when 5, 10 or 20 documents have been seen. Thus, it is common to evaluate each query's ranking using the precision at the top k positions (typically $P@5$, $P@10$ and $P@20$):

$$P@k(q_i) = \frac{\text{\# of relevant documents in the top } k \text{ positions}}{k} \tag{2.8}$$

Mean Average Precision (MAP)

To evaluate the overall quality of a ranking method, it is easier to use a single value summarizing the precision obtained for all queries. The Mean Average Precision gives us the mean of the Average Precision (AP) for each of the m queries evaluated. The AP for each query is defined as:

$$AP(q_i) = \frac{\sum_{k=1}^{n} P@k(q_i) \times l_k}{\text{\#of relevant documents}} \tag{2.9}$$

Where n is the total number of documents associated with query q_i and l_k is 1 if the document at the k^{th} position in the ranking is relevant or 0 if it is not relevant. With the AP for each query, we can calculate the MAP for all m queries:

$$MAP = \frac{1}{m} \sum_{i=1}^{m} AP(q_i) \tag{2.10}$$

Mean Reciprocal Rank (MRR)

In certain cases, we are interested only in the first correct answer to a query. For instance, in web search engines, certain queries are "navigational": the user is looking for a specific web page and, thus, interested in one particular relevant result. In such situations, we want a measure that favors results where the first relevant answer is higher in the resulting ranking. For a set of queries $Q = \{q_1, ..., q_m\}$, we define $rank_i$ as the position of the first relevant document returned for a query $q_i \in Q$, then the MRR of the rankings of the documents returned by the queries in Q is given by:

$$MRR = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{rank_i} \tag{2.11}$$

Normalized Discounted Cumulative Gain (NDCG)

The previously described metrics work only for binary judgments (i.e. 0 = relevant, 1 = not relevant). If the document assessment is done using levels of relevance (e.g. 0 = not relevant, 1 = somewhat relevant, 2 = relevant, 3 = very relevant), then a ranking with some "very relevant" documents at the top may be better than another ranking with more "somewhat relevant" documents at the top. Another drawback is that the metrics presented do not differentiate between a relevant document at the very top of the ranking and another at the very bottom. To address these issues, the Discounted Cumulative Gain ($DCG@k(q_i)$) for a query q_i is a metric that uses cumulative gain vectors to allow for multiple relevance levels (i.e. 0, 1, 2, etc.) and a discount factor that reduces the impact of the gain as it moves up in the ranking. The DCG of a query q_i at position k is given by

$$DCG@k(q_i) = \sum_{r=1}^{k} \frac{2^{l_r} - 1}{\log(r+1)} \tag{2.12}$$

Where l_r is the relevance level for the document at position r in the ranking. To be able to compare the DCG of several queries, we need to normalize it using the ideal

DCG for that query (at the specified position k). The ideal DCG, or IDCG, is obtained by ordering the documents returned for a query in descending order of relevance, thus obtaining the maximum DCG possible up to the selected position k.

$$NDCG@k(q_i) = \frac{DCG@k(q_i)}{IDCG@k(q_i)} \tag{2.13}$$

The NDCG@k for all queries can be averaged to obtain a single measure of the quality of a ranking function

$$NDCG@k = \frac{1}{m}\sum_{i=1}^{m} NDCG@k(q_i) \tag{2.14}$$

Like with precision, the NDCG is usually measured at positions 5, 10 or 20 (i.e. NDCG@5, NDCG@10, NDCG@20).

2.3 Benchmarking Datasets

Another important element of the Evaluation Framework is the availability of benchmarking datasets. A newly proposed ranking model can then be tested using a well known benchmarking dataset and the results obtained (using various metrics, including some of those described above) can be compared to the published results of other research groups to gauge the quality of the new method. Well established models such as BM25 or the vector model can be run using these datasets, providing a common *baseline* for comparison. To this end, the Text REtrieval Conferences[1] (TREC) were established in the early nineties. The conference is held yearly and is dedicated to experimentation with large test collections. Research groups participate in the conference and run experiments comparing their retrieval systems. TREC publishes several document and web collections of different sizes and purposes that can be used by researchers to evaluate their IR methods.

As we will see in Section 4, there is a collection of benchmarking datasets designed specifically for learning to rank (which we use in all the experiments presented in this work). It is not published by TREC, but it uses TREC collections to produce datasets specifically designed for learning to rank methods.

[1]http://trec.nist.gov

3 Learning to Rank

Although the classic approaches to ranking that we discussed in section 1.2 usually yield very good results, ranking remains a very difficult task. A ranking algorithm is supposed to predict which documents the user who posed the query will consider relevant. Two users with the same information need may find different documents relevant (or not relevant) or pose very different queries. Queries may be ill formed or too short, yielding too many or too few results. As a consequence, an IR system should implement an algorithm that tries to rank documents in such a way as to approximate the opinion of relevance of a large fraction of its users and that are obtained through the variety of queries posed to the system.

Some of the classic methods described in the previous section can be tuned to improve their performance in specific document collections; parameters k_1 and b in the BM25 equation (2.5), for instance, can be adjusted to better fit a certain collection. Tuning these parameters is usually done experimentally using a validation set, where part of the collection is used to evaluate different parameter values. But it is far from simple to achieve good tuning by experimentation alone. Machine Learning algorithms can be used to automate the parameter tuning process, as proposed, for example, by Almeida et al. [2007]. Although parameter tuning is one of the applications of ML in Information Retrieval, ML techniques can be directly applied to the ranking problem itself, yielding very good results. ML methods applied to ranking are known as Learning to Rank (or L2R, for short).

L2R usually involves the use of *supervised learning* techniques where a ranking model or function is derived from *training data* containing documents returned for a query and ranked by a human assessor. This ranking model can then be used to rank documents for a new query (i.e. not yet seen) posed by a user. A L2R system is typically used to enhance an existing IR system; its function is not to *retrieve* documents from the collection, but to *reorder* the list of documents returned by the IR system as a response to a user query. Figure 2.2 shows a simplified diagram describing how a L2R method relates to an existing IR infrastructure. The diagram contains two distinct phases of the L2R process:

1. **Training data gathering**: To be able to build a L2R model, the learning system needs to have training data available. To gather the training data, queries are posed to the IR system, which returns ranked lists of documents satisfying the queries (e.g. using a classic IR ranking model). Human assessors then evaluate the top k documents indicating, for instance, which ones are relevant to the query

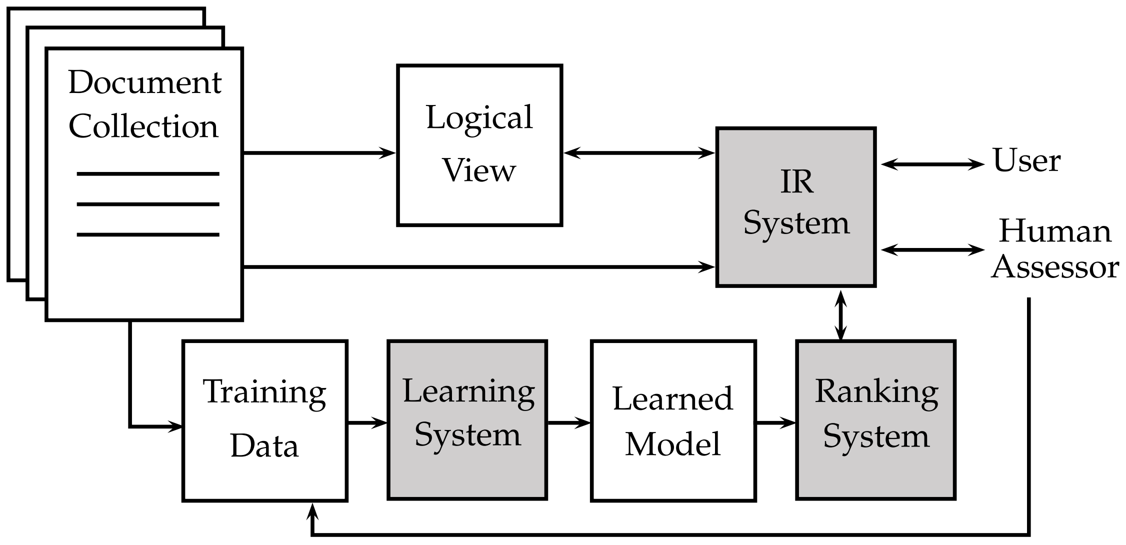

Figure 2.2. Learning to Rank Framework

(this process is called "labeling"). The labeled documents are then inserted into the training data set. The training data, like the logical view of the documents used by the IR system, is a *representation* of document entities, as we will see below. This first phase is performed once or from time to time (i.e. offline). Once there is training data available the learning system can create a L2R model that can be used to rank documents returned by new queries (there is a whole process of evaluating and fine-tuning the L2R model that does not appear in the diagram but that we will analyze in detail throughout this work).

2. **Ranking document lists returned by user queries**: Once we have an effective learned model in place, the ranking system can rank new document lists. As in the classic IR framework, a user poses a query to the system, which uses the logical view to obtain the documents that are relevant to the query. The documents of this list are then passed by the IR system to the ranking system (in the same "format" or representation used for the training data) which ranks the documents, returning this reordered list back to the IR system to be processed and presented to the user.

The training data, similarly to the "logical view" used by the IR system, is a representation of data instances. Instead of using weights to associate index terms (i.e. the vocabulary) with documents, it uses *feature vectors* reflecting the relevance of documents to queries. These features are numerical properties of the query-document pairs, such as the TF-IDF of query terms or the BM25 score of a document for a given query. Document-only features can also be used, such as the HITS or PageRank scores of a web page (see section 4 for examples of features used in the LETOR 3.0 web datasets). Just like in any supervised learning task, the idea is to "teach" the learning algorithm the relationship between the feature values and the label/ranking so that the learned model produced from the training data may be used to predict the relevance or ranking of documents in relation to a new query posed by the user. Observe that a data instance in L2R is always a feature vector of a document *in relation* to a query. That is, the same actual document can appear several times in the training data but with very different feature values if it was returned by more than one query. For our purposes, these are different instances even if they are related to the same document in the collection. Throughout this work we interchangeably use the terms instance, sample, example and document to refer to a query-document feature vector.

Liu [2009] identifies three categories of learning to rank algorithms: pointwise, pairwise and listwise. These categories have different input, output and hypothesis spaces, and employ different loss functions. Here we briefly describe each category with emphasis in the input and output spaces which directly affect how the training data is produced. For a comprehensive analysis of these categories, see Liu [2009].

Pointwise

Pointwise L2R algorithms consider each document independently of each other. That is, the input space contains the feature vector of each single document. The output space of pointwise methods is a relevance degree or relevance level. This means that for this type of methods, the label provided by the human assessor is a discrete relevance level; it could be binary (i.e. 0 = not relevant, 1 = relevant), or contain several levels (e.g. 0 = not relevant, 1 = somewhat relevant, 2 = relevant, 3 = very relevant). In this scenario, the L2R algorithm may use, for instance, classification or regression to predict the relevance level of documents returned by new queries. A scoring function is used to output a score for each document, allowing for the easy ranking of the documents. The association rule-based algorithm used as the basis for our active learning method and presented in section 5 is an example of a pointwise L2R method.

Pairwise

Pairwise methods use pairs of documents to learn pairwise preferences. That is, instead of considering each document separately, these methods look at the relative relevance of two documents; e.g. document d_i is more relevant than document d_j (or $d_i \succ d_j$). Thus, the input space contains pairs of document feature vectors, while the output space contains a pairwise preference indicating the order of relevance of the documents in the pair (e.g {1, -1} for the pair d_i, d_j). Ideally, the labeling process should provide assessment pairs indicating the relative relevance of all document pairs. This makes the human assessment process more laborious, as we will see in Chapter 3, Section 1. It is possible to use relevance levels (binary or otherwise) to create pairwise labels by defining the label as $y_{i,j} = 2 \times I_{\{l_i \succ l_j\}} - 1$, where $I_{\{A\}}$ is an indicator function that returns 1 if A is true and 0 otherwise. Notice that in this case only comparisons between documents with different relevance levels yield useful results. Similarly, if the judgment is given as a total order π_l (see the listwise approach below), we can define $y_{i,j} = 2 \times I_{\pi_l(i) < \pi_l(j)} - 1$. Two well known instances of the pairwise method are SVMRank and RankBoost as we shall see in Section 6.

Listwise

In the listwise approach, the input space contains all documents associated with a query. The output space can take two forms (depending on the listwise algorithm's formulation): of ranked lists or permutations of documents (i.e. a total order of the documents); or relevance levels similarly to the pointwise approach. In the former case, the labels are given as a total order (i.e. a ranking of the documents of a query) π_l and the ground truth is defined as $\pi_y = \pi_l$. If the training data contains relevance degree labels, it is possible to derive π_y by using an equivalent permutation set of the form $\Omega_y = \{\pi_y | u < v, \text{if } l_{\pi_y^{-1}(u)} \succ l_{\pi_y^{-1}(v)}\}$. Similarly, if the labels are pairwise preferences, we can use $\Omega_y = \{\pi_y | u < v, \text{if } l_{\pi_y^{-1}(u), \pi_y^{-1}(v)} = 1\}$. One advantage of using a listwise method with π_y as the ground truth label is that the output space used in the learning process is exactly the output space of the ranking task.

4 Benchmarking Datasets for L2R

With the advent of many learning to rank techniques, it becomes important to be able to compare the different methods proposed by researchers. The task can be made much easier if the proposed methods are tested against common benchmarking datasets. A widely known collection of datasets specifically tailored for L2R is the LEarning TO Rank (LETOR) collection[2].

As we will see in Chapters 4 and 5, we did extensive experimentation using the Learning to Rank (LETOR) benchmark datasets version 3.0. Here we briefly discuss the main characteristics of these datasets.

[2] http://research.microsoft.com/en-us/um/beijing/projects/letor/

Table 2.1. LETOR 3.0 Web Datasets' Characteristics

	# Qry	**# Qry TR**	**TR Size**	**TS Size**	**D/qry**	**R%**	**R/Q**
TD2003	50	30	29,434	9,812	981	1.52	14.98
TD2004	75	45	44,487	14,829	988	2.92	28.92
NP2003	150	90	89,194	29,731	991	0.10	1.01
NP2004	75	45	44,300	14,767	984	0.17	1.64
HP2003	150	90	88,563	29,521	984	0.12	1.22
HP2004	75	45	44,645	14,882	992	0.13	1.25

LETOR 3.0 is composed of 6 web datasets plus the OHSUMED corpus. The web datasets contain labeled instances selected from web pages obtained from a 2002 crawl of the .gov TLD. These collections are separated in three search tasks: topic distillation (TD), homepage finding (HP) and named page finding (NP) and contain 2 sets each (namely, TREC 2003 and 2004). The TD datasets are basically comprised of "informational" queries, i.e., queries whose target are documents containing information about a specific topic, theme or entity. The HP and NP datasets are focused on "navigational" queries whose target is usually a single specific Web page.

These collections contain instances represented by 64 features for the top 1,000 documents returned for a specific set of queries using the BM25 model (see Qin et al. [2010b] for a detailed description of how these datasets were created). These datasets use a binary relevance judgment indicating whether a document is or is not relevant to a given query. We evaluate our method on all the largest, more diverse, LETOR 3.0 web datasets (namely, TD2003, TD2004, HP2003, HP2004, NP2003 and NP2004). In all experiments we have used the query-based normalized versions as suggested by the producers of the LETOR benchmarking datasets. We also use 5-fold cross validation for all results reported as well as the evaluation script provided in the LETOR package to generate the final MAP and NDCG metrics.

Table 2.1 summarizes the characteristics of each dataset. The "# Qry" column contains the total number of queries in the dataset (i.e. in each fold we have this many queries in the training, validation and test sets combined). "# Qry TR" indicates how many queries are there in the training set in each fold. "TR Size" and "TS Size" show the number of instances in the training and test sets, respectively. "D/qry" shows the number of documents per query, the "R%" column indicates the percentage of relevant documents in the dataset and the "R/Q" show the average of relevant documents per query for the whole dataset. "TR Size", "TS Size" and "D/qry" are averages accross the 5 folds.

Features

The 64 features appearing in the LETOR 3.0 web datasets are either related to the query+document or to the document alone. In the former group we have, for instance, the TF of the body, anchor, title, URL and the whole document (features 1 to 5). There's also features for the IDF and the TF-IDF (6 to 15). Then there are features with values for classic IR models such as BM25 (21 to 25) and several for various language models as proposed by Zhai and Lafferty [2004] (26 to 40 and 62 to 64). There are also sitemap (Qin et al. [2005]) and hyperlink (Shakery and Zhai [2003]) based propagation features (41, 42 and 43 to 48, respectively). Features related to the document alone include various document lengths (body, title, URL, etc.; 16 to 20) and web related features (49 to 60), containing values such as the number of inlinks and outlinks (56, 57), the length of the page's URL (59), the number of child pages (60), PageRank (Page et al. [1999]) and HITS (Kleinberg [1999]).

5 Learning to Rank Using Association Rules: RLR

Recently, a new pointwise learning to rank method was proposed by Veloso et al. [2008]. This Rule-based Learning to Rank algorithm (that we henceforth refer to as RLR) is a supervised method that derives association rules from the training data at query time (i.e. on-demand). We used their method as a basis to develop the active learning algorithm we present in Chapter 4. RLR is very effective as can be seen in the results presented in Veloso et al. [2008] for some of the LETOR 2.0 datasets. We also discuss the results obtained by RLR on the LETOR 3.0 datasets in Chapter 4, Section 3 (see tables 4.1 and 4.2) and use them as a baseline to evaluate our active method's effectiveness. In this section we briefly describe RLR. The reader is encouraged to read the aforementioned work for a more detailed description.

5.1 Rule-based Learning to Rank

For the purpose of describing RLR, the task of learning to rank is defined as follows. We have as input the *training data* (referred to as $\mathcal{D}$), which consists of a set of records of the form $< q, d, r >$, where q is a query, d is a document (represented as a list of m feature-values or $\{f_1, f_2, \ldots, f_m\}$), and r is the *relevance* of d to q. Features include BM25, Page Rank, and many other document and query-document properties. The relevance of a document draws its values from a discrete set of possibilities $\{r_0$, r_1, …, $r_k\}$ (e.g. 0: not relevant, 1: somewhat relevant, 2: very relevant). The training data is used to build functions relating features of the documents to their corresponding relevance. The *test set* (referred to as $\mathcal{T}$) consists of records $< q, d, ? >$ for which only the query q and the document d are known, while the relevance of d to q is unknown. Ranking functions obtained from $\mathcal{D}$ are used to estimate the relevance of such documents to the corresponding queries.

Ranking functions exploit the relationship between document features and relevance levels. This relationship may be represented by association rules. We denote as $\mathcal{R}$ a rule-set composed of rules of the form $\{f_j \wedge \ldots \wedge f_l \xrightarrow{\theta} r_i\}$. These rules can contain any mixture of the available features in the antecedent and a relevance level in the consequent. The strength of the association between antecedent and consequent is measured by a statistic, θ, which is known as *confidence* (see Agrawal et al. [1993]) and is simply the conditional probability of the consequent given the antecedent. RLR extracts rules from $\mathcal{D}$ on a demand-driven basis and then combine these rules in order to estimate the relevance of each document in $\mathcal{T}$.

5.2 Demand-Driven Rule Extraction

The search space for rules is huge, and thus, computational cost restrictions must be imposed during rule extraction. Typically, a minimum support threshold (σ_{min}) is employed in order to select frequent rules (i.e., rules occurring at least σ_{min} times in $\mathcal{D}$) from which the ranking function is produced. This strategy, although simple, has some problems. If σ_{min} is set too low, a large number of rules will be extracted from $\mathcal{D}$, and often most of these rules are useless for estimating the relevance of documents in $\mathcal{T}$ (a rule $\{\mathcal{X} \rightarrow r_i\}$ is only useful to estimate the relevance of document $d \in \mathcal{T}$ if the set of features $\mathcal{X} \subseteq d$, otherwise the rule is meaningless to d). On the other hand, if σ_{min} is set too high, some important rules will not be included in $\mathcal{R}$, causing problems if some documents in $\mathcal{T}$ contain rare features (i.e., features occurring less than σ_{min} times in $\mathcal{D}$). Usually, there is no optimal value for σ_{min}, that is, there is no single value that ensures that only useful rules are included in $\mathcal{R}$, while at the same time important rules are not missed. The method to be proposed next deals with this problem by extracting rules on a demand-driven basis.

Demand-driven rule extraction is delayed until a set of documents is retrieved for a given query in $\mathcal{T}$. Then, each individual document d in $\mathcal{T}$ is used as a filter to remove irrelevant features and examples from $\mathcal{D}$. This process produces a projected training data, $\mathcal{D}_d$, which is obtained after removing all feature-values not present in d. Then, a specific rule-set, $\mathcal{R}_d$ extracted from $\mathcal{D}_d$, is produced for each document d in $\mathcal{T}$.

Lemma 1: All rules extracted from $\mathcal{D}_d$ (i.e., $\mathcal{R}_d$) are useful to estimate r.

Proof: Since all examples in $\mathcal{D}_d$ contain only feature-values that are present in d, the existence of a rule $\{\mathcal{X} \rightarrow r_i\} \in \mathcal{R}_d$, such that $\mathcal{X} \not\subseteq d$, is impossible. ■

Theorem 1: The number of rules extracted from $\mathcal{D}$ depends solely on the number of features in $\mathcal{D}_d$, no matter the value of σ_{min}.

Proof: If an arbitrary document $d \in \mathcal{T}$ contains at most l features then any rule matching d can have at most l feature-values in its antecedent. That is, for any rule $\{\mathcal{X} \rightarrow r_i\}$, such that $\mathcal{X} \subseteq d$, $|\mathcal{X}| \leq l$. Consequently, for $\sigma_{min} \approx 0$, the number of possible rules matching d is $k \times (l + \binom{l}{2} + \ldots + \binom{l}{l}) = O(2^l)$, where k is the number of distinct relevances. Thus, the number of rules extracted for all documents in $\mathcal{T}$ is $O(|\mathcal{T}| \times 2^l)$. ■

5.3 Relevance Estimation

In order to estimate the relevance of a document d, it is necessary to combine all rules in $\mathcal{R}_d$. Our strategy is to interpret $\mathcal{R}_d$ as a poll, in which each rule $\{\mathcal{X} \xrightarrow{\theta} r_i\} \in \mathcal{R}_d$ is a vote given by a set of features $\mathcal{X}$ for relevance level r_i. Votes have different weights, depending on the strength of the association they represent (i.e., θ). The weighted votes for relevance level r_i are summed and then averaged (by the total number of rules in $\mathcal{R}_d$ that predict relevance level r_i), forming the score associated with relevance r_i for document d, as shown in Equation 2.15 (where $\theta(\mathcal{X} \rightarrow r_i)$ is the value θ assumes for rule $\{\mathcal{X} \rightarrow r_i\}$ and $\mathcal{R}_d^{r_i}$ is the set of rules derived for document d that predict relevance level r_i):

$$s(d, r_i) = \frac{\sum \theta(\mathcal{X} \rightarrow r_i)}{\mid \mathcal{R}_d^{r_i} \mid}, \text{ where } \mathcal{X} \subseteq d \tag{2.15}$$

Therefore, for a document d, the score associated with relevance r_i is given by the average θ values of the rules in $\mathcal{R}_d$ predicting r_i. The likelihood of d having a relevance level r_i is obtained by normalizing the scores, as expressed by $\hat{p}(r_i|d)$, shown in Equation 2.16:

$$\hat{p}(r_i|d) = \frac{s(d, r_i)}{\sum\limits_{j=0}^{k} s(d, r_j)} \tag{2.16}$$

Algorithm 1 Rule-based Learning to Rank (RLR)

```
Require: The training data $\mathcal{D}$, test set $\mathcal{T}$, and $\sigma_{min}$ ($\approx 0$)
Ensure: $rank(d)$
 1: for all pair $(d, q) \in \mathcal{T}$ do
 2:     $\mathcal{D}_d \Leftarrow \mathcal{D}$ projected according to $d$
 3:     $\mathcal{R}_d \Leftarrow$ rules extracted from $\mathcal{D}_d \mid \sigma \geq \sigma_{min}$
 4:     for all $i \mid 0 \leq i \leq k$ do
 5:         $s(d, r_i) \Leftarrow \frac{\sum \theta(\mathcal{X} \rightarrow r_i)}{|\mathcal{R}_d^{r_i}|}$
 6:     end for
 7:     $rank(d) \Leftarrow 0$
 8:     for all $i \mid 0 \leq i \leq k$ do
 9:         $rank(d) \Leftarrow rank(d) + r_i \times \hat{p}(r_i|d)$
10:     end for
11: end for
```

Finally, the relevance of document d is estimated by a linear combination of the likelihoods associated with each relevance level, as expressed by the ranking function $rank(d)$, which is shown in Equation 2.17:

$$rank(d) = \sum_{i=0}^{k} \big(r_i \times \hat{p}(r_i|d)\big) \tag{2.17}$$

The value of $rank(d)$ is an estimate of the true relevance of document d (i.e., r) using $\hat{p}(r_i|d)$. This estimate ranges from r_0 to r_k, where r_0 is the lowest relevance and r_k is the highest one. Relevance estimates are used to produce ranked lists of documents. All steps of RLR are depicted in Algorithm 1.

6 Other Supervised Learning to Rank Algorithms

Over the years, many different L2R algorithms have been proposed. Most of the pointwise methods have been derived and adapted from established classification or regression algorithms. This is the case of RLR, which is based on classification ML techniques presented in Veloso and Meira, Jr. [2011]. Another example of a pointwise method based on classification is McRank, introduced by Li et al. [2007]. Ordinal regression methods have also been proposed, such as the perceptron-based PRanking (see Crammer and Singer [2001]) or the SVM-based method proposed by Shashua and Levin [2003].

Two very famous pairwise methods, SVMRank and RankBoost, are detailed below. Other pairwise methods use neural networks and gradient descent optimization to minimize pairwise loss functions, such as the *cross entropy loss* for RankNet (Burges et al. [2005]) and the *fidelity loss* for FRank (Tsai et al. [2007]).

Given the characteristics of the input space used by the listwise approach to L2R (i.e. the use of total or partial ordering permutations), several algorithms in this category try to directly optimize ranking metrics such as MAP or NDCG. But since these metrics, being position based, are non-continuous and non-differentiable (see Robertson and Zaragoza [2007]; Xu et al. [2008]), some methods choose to optimize approximations of these metrics. This is the case with SoftRank (Taylor et al. [2008]) and AppRank (see Qin et al. [2010a]). Other algorithms try to optimize restricted or bounded IR measures, such as SVMMap (Yue et al. [2007]) and PermuRank (Xu et al. [2008]). Finally, some methods use optimization techniques that are able to optimize complex objectives which is the case of AdaRank (Xu and Li [2007]), for instance.

In the following sections we briefly describe SVMRank and RankBoost. These are the algorithms that we use in the Query-By-Committee (QBC) strategy delineated in Chapter 5 (along with RLR), and we believe the reader may benefit from a quick overview of the inner workings of these methods.

6.1 SVMRank

Joachims [2002] proposed a very interesting method called SVMRank that uses web search clickthrough data to train an SVM-based L2R algorithm. Joachims arguments that user clicks on query results provide a *relative* relevance judgment that can be modeled as a pairwise label. That is, given a query q, the ranking presented as a result to this query r and the list of the documents that the user clicked c (i.e. a triplet (q, r, c)), we can generate a set of pairwise judgments and train an SVM using them. Let's say that, presented with a list of documents retrieved as a response to a query, the user clicked on documents 1, 3 and 7. The fact that document 2 was skipped (i.e. not clicked) is an indication that the user found documents 3 and 7 to be more relevant than document 2 (assuming the user scans the ranked list from top to bottom and that the content snippets are equally informative). Although we cannot say how relevant document 3 is on an *absolute* scale, we can say that document 3 is, in this user's assessment, more relevant to q than document 2 (i.e. $d_3 \succ d_2$, where $d_3, d_2 \in r$). Similarly, we can infer, from this particular ranking, that document 7 is more relevant to the query then documents 6, 5, 4 and 2 ($d_7 \succ d_6, d_7 \succ d_5, d_7 \succ d_4, d_7 \succ d_2$).

Instead of trying to minimize training error, as is commonly done in classification, Joachims proposes maximizing the Kendall's τ value of the rankings generated by the learned function and the ground truth rankings of the training set. Let's call $f(q)$ the learned function, $r_{f(q_i)}$ the ranking obtained by using the function on the documents of query q_i and r_i^* the optimal ordering for this query. Given these two rankings and the m documents associated to query q_i:

$$\tau(r_{f(q_i)}, r_i^*) = \frac{P-Q}{P+Q} = 1 - \frac{2Q}{\binom{m}{2}} \tag{2.18}$$

where P is the number of concordant pairs in the rankings and Q is the number of discordant pairs. Notice that $\tau(r_{f(q_i)}, r_i^*) = 1$ if the rankings are identical and -1 if they are completely different. Also notice that $P + Q = \binom{m}{2}$, yielding the alternative formulation to the right.

Given n queries and their corresponding target rankings $(q_1, r_1^*), ..., (q_n, r_n^*)$, the learner must select a ranking function f from a family of ranking functions F that maximizes the empirical τ:

$$\tau(f) = \frac{1}{n}\sum_{i=1}^{n} \tau(r_{f(q_i)}, r_i^*) \tag{2.19}$$

Each document d returned by a query q is represented as a feature vector $\vec{x}$ of values correlating the query and the document (similar to the features used in the LETOR web datasets and discussed in Section 4). Consider the class of linear ranking functions $f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle$ that satisfies $\vec{x}_i \succ \vec{x}_j \Leftrightarrow \langle \vec{w}, \vec{x}_i \rangle > \langle \vec{w}, \vec{x}_j \rangle$ where $\vec{w}$ is the weight vector adjusted by learning. In Figure 2.3 we see a two-dimensional example of how a weight vectors determine the ordering of four points. The points are ordered by their projection onto $\vec{w}$ or, equivalently, by their signed distance to a hyperplane with normal vector $\vec{w}$. Thus, vector $\vec{w}_1$, orders the points as 1, 2, 3, 4 whilst $\vec{w}_2$ determines an order of 2, 3, 1, 4.

Instead of maximizing Equation 2.19 directly, it is equivalent to minimize the number Q of discordant pairs in 2.18. For the class of ranking functions $f(\vec{x})$ this is equivalent to finding a weight vector $\vec{w}$ so that the maximum number of the following inequalities is fulfilled:

$$\begin{gathered} \forall(\vec{x}_i, \vec{x}_j) \in r_1^* : \langle \vec{w}, \vec{x}_i \rangle > \langle \vec{w}, \vec{x}_j \rangle \\ \dots \\ \forall(\vec{x}_i, \vec{x}_j) \in r_n^* : \langle \vec{w}, \vec{x}_i \rangle > \langle \vec{w}, \vec{x}_j \rangle \end{gathered} \tag{2.20}$$

Unfortunately, this problem is NP-hard. Instead, Joachims uses the soft-margin SVM model proposed by Vapnik [1995] as an approximate solution to the problem of

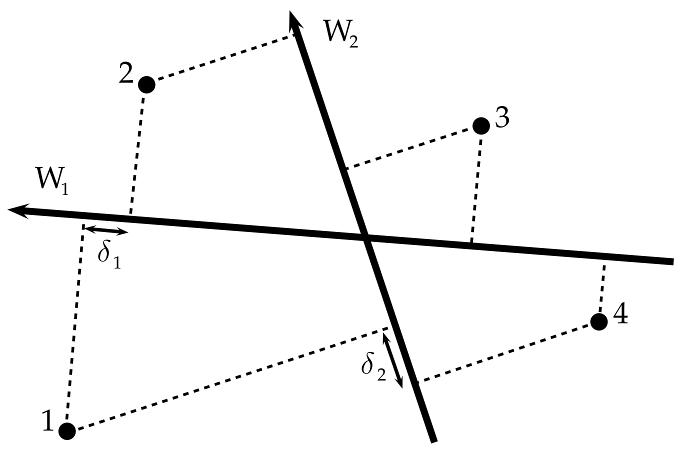

Figure 2.3. SVMRank: two-dimensional example of how two possible weight vectors order four documents

finding f, formulating it as a quadratic optimization problem:

$$\min_{\vec{w}} \frac{1}{2} \parallel \vec{w} \parallel^2 + C \sum \xi_{ijk} \tag{2.21}$$

subject to:

$$\begin{aligned} \forall(\vec{x}_i, \vec{x}_j) \in r_1^* &: \langle \vec{w}, \vec{x}_i \rangle \geq \langle \vec{w}, \vec{x}_j \rangle + 1 - \xi_{ij1},\ \xi_{ij1} \geq 0 \\ &\dots \\ \forall(\vec{x}_i, \vec{x}_j) \in r_n^* &: \langle \vec{w}, \vec{x}_i \rangle \geq \langle \vec{w}, \vec{x}_j \rangle + 1 - \xi_{ijn},\ \xi_{ijn} \geq 0 \end{aligned} \tag{2.22}$$

where C is a parameter that allows trading off margin size for training error. Geometrically, the SVM margin δ is the distance between the closest two projections as seen if Figure 2.3.

By using the pairwise comparisons provided by the clickthrough data as a training set, SVMRank is able to find the $\vec{w}^*$ that minimizes 2.21 given the constraints 2.22. After that, this model (i.e. $\vec{w}^*$) can be used to rank documents for a new query q by sorting them by their value $\langle \vec{w}^*, \vec{x}_k \rangle$.

6.2 RankBoost

Another pairwise method proposed by Freund et al. [2003] is RankBoost. Based on the well-know classification method AdaBoost (developed by Freund and Schapire [1997]) RankBoost uses a boosting approach to ranking. The essential idea of boosting methods is to combine the results of what are called *weak learners* into a single accurate (or "strong") learned function. This is done in a round-based fashion where at each round the same weak learner is used but the input to the weak learner is modified by increasing the weight or importance of hard to determine pairwise preferences. Thus, as the rounds progress, more importance is given to correctly learning the pairwise preference on the pairs whose relative ranking is hardest to determine. In other words, pairs that are difficult to rank are "boosted" by increasing weights while correctly ranked pairs have their weights reduced at each round.

More formally, given the training set of document instances X, each document $x \in X$ is composed of feature-values as before. A feedback function Φ can be used on all pairs of instances to provide the pairwise preference. Thus, $\Phi(x_i, x_j)$ returns 1 if $x_i \succ x_j$, -1 if $x_i \prec x_j$ or 0 if there is no preference (e.g. for relevance degrees, x_i and x_j have the same relevance label). The goal of the learner is to output a function $H : X \rightarrow \mathbb{R}$ such that $x_i \succ x_j \Leftrightarrow H(x_i) > H(x_j)$. To achieve this goal, the learner uses a distribution D that is updated at each round by the weak learner. This distribution is defined as:

$$D(x_i, x_j) = c \times max\{0, \Phi(x_i, x_j)\} \tag{2.23}$$

This is done in order to eliminate the redundant information provided by negative values of Φ (i.e. if $\Phi(x_i, x_j) = -1$, then this information is already present in the opposite pair $\Phi(x_j, x_i) = 1$ and thus it is set to zero). c is a positive constant chosen so that

$$\sum_{x_i, x_j} D(x_i, x_j) = 1 \tag{2.24}$$

Rankboost defines *crucial pairs* as those pairs where $\Phi(x_i, x_j) > 0$. These are the pairs whose weights are updated at each round (i.e. "boosted" if they are incorrectly ordered by the weak learner and reduced if correctly ordered). Thus the algorithm is designed to find an H with a small number of crucial pair misorderings on the training set or, in other words, to minimize the pairwise ranking loss.

At each round t, the algorithm passes a distribution D_t to the weak learner so that it can produce a *weak hypothesis* of the form $h_t : X \rightarrow \mathbb{R}$. The distribution is updated at each round by doing:

$$D_{t+1}(x_i, x_j) = \frac{D_t(x_i, x_j)\ exp(\alpha_t(h_t(x_i) - h_t(x_j)))}{Z_t} \tag{2.25}$$

where Z_t is a normalization factor chosen so that D_{t+1} will be a distribution and $\alpha \in \mathbb{R}$ is a parameter chosen at each round (more on Z_t and α below).

We can run the algorithm for as many rounds as desired. Once it is finished, it outputs the final hypothesis as a weighted sum of the weak hypotheses produced at each round:

$$H(x) = \sum_{t=1}^{T} \alpha_t \times h_t(x) \tag{2.26}$$

In order to produce a combined hypothesis with low ranking loss, at each round α_t should be chosen and the weak learner should build h_t so as to minimize

$$Z_t = \sum_{x_i, x_j} D_t(x_i, x_j)\ exp(\alpha_t(h_t(x_i) - h_t(x_j))) \tag{2.27}$$

Freund et al. [2003] provides three example methods on how to approximate this minimization problem efficiently. We do not delve into the details of these methods here. The reader is encouraged to consult the aforementioned work for the details.

The Weak Learner

RankBoost treats the weak learner as a black box. Different weak learners can be implemented and used, although they should take into consideration Equation 2.27 when building h_t. One possible learner for the ranking problem is:

$$h(x) = \begin{cases} 1 & \text{if } f_i^x > \theta \\ 0 & \text{if } f_i^x \leq \theta \\ s_{def} & \text{if } f_i^x = 0 \end{cases} \tag{2.28}$$

where f_i^x is the value of the i^{th} feature of document x, and $0 \leq s_{def} \leq 1$ is a default value assigned to $h(x)$ if, for some x, the feature value for the i^{th} feature is null (or zero, assuming that feature-values are normalized and zero indicates a null value such as is the case with the query level normalized LETOR datasets). Here (at each round), the weak learner must use the distribution D passed on to it and the labels provided by $\Phi(x_i, x_j)$ to determine which feature f_i will be used and what are the best values for θ and s_{def} in order to learn the weak hypothesis $h(x)$. Freund et al. [2003] details an efficient algorithm for the weak learner above that also approximately minimizes 2.27.

7 Summary

Ranking is an essential functionality of many systems. Although classic Information Retrieval models and systems have been designed to provide ranked results, in the last decade Machine Learning techniques have been successfully applied to the ranking problem. In practice, learning to rank systems are used to enhance an IR system by re-ranking the results returned to the user according to a learned model or function.

In this chapter we have seen how this L2R framework fits into the general picture of an IR system and how we can evaluate the quality of new ranking methods, be them classic IR models or ML-based. We have also described in some detail three very different L2R methods: RLR, SVMRank and RankBoost. These descriptions will be useful when we introduce a novel rule-based active learning technique in Chapter 4 and then proceed to expand our method using a QBC strategy in Chapter 5.

Chapter 3

Active Learning

1 The Labeling Effort

To be able to build a learning to rank model it is necessary to provide the learning algorithm with training data. That is, query-document pairs represented as feature vectors and labeled by human assessors. As we have seen in the last chapter, a supervised learning to rank method uses the training data to produce a model that is later used for ranking new queries (in the case of RLR, there is no model, but the training data is used to rank a new query on demand). Obtaining the training data is usually a labor-intensive task, with several human assessors reviewing the documents retrieved by the IR system for the specified training building queries.

In many cases, the labeling task requires the assessors to decide whether a given document is or is not relevant to the query (i.e. $\{0, 1\}$). A common extension is to use relevance levels, indicating if a document is more or less relevant to the query (e.g. $\{0, 1, 2, 3\}$). In either case, it is straightforward to use the training data obtained through the assessment process to train pointwise L2R algorithms. As we have pointed out, it is also possible to modify relevance labels for use with a pairwise algorithm by defining the label as $y_{i,j} = 2 \times I_{\{l_i \succ l_j\}} - 1$. But this transformation is not ideal. The resulting pairwise comparisons are only useful when the compared documents have different relevance levels. The number of instance pairs is, in the worst case, quadratic to the number of documents and depends on the number of relevant documents obtained for the query. Thus the difference in the number of pairs between queries can be very large, which could lead to a L2R model dominated by the queries with large numbers of document pairs. Cao et al. [2006] analyses this problem in the context of SVM ranking and proposes query-level normalization on the pairwise loss function in order to minimize this effect.

The same holds true for the listwise methods which expect the ground truth to be given as the total order of the documents. It is possible to convert relevance level judgments using an equivalent permutation set. But the resulting permutations obtained may not be the ground truth permutations, since we cannot determine the order of documents with the same relevance level.

Ideally, the assessment strategy should reflect the type of learning algorithm used and provide labels that maximize the method's effectiveness. This makes the assessment process even harder than it already is, since pairwise labeling and total or partial relevance ordering are more difficult to perform. Independently of how the training data is constructed, it is costly and laborious to produce any amount of it.

Although in this work we perform experimentation using binary relevance datasets (and, therefore, we assume that the human assessment of selected instances is also binary), the active learning techniques presented here could be easily extended to allow for pairwise or partial ordering relevance assessments.

2 Reducing the Labeling Effort: Active Learning

Active learning techniques for L2R have been proposed to help deal with the labeling effort problem (see, for instance, Silva et al. [2011]; Cai et al. [2011]; Donmez and Carbonell [2009, 2008]; Long et al. [2010]; Radlinski and Joachims [2007]; Yu [2005]). The motivation behind active learning is that it may be possible to achieve highly effective learning functions by carefully selecting and labeling instances that are "informative" to the learning algorithm. Using active learning, we can reduce the cost of producing training sets for rank learning algorithms and even improve the effectiveness of the learned functions by avoiding adding "noisy" instances to the training sets. Furthermore, human annotators can spend more time analyzing the relevance of each actively selected instance, possibly producing better training data (see Geng et al. [2011] for techniques on assessing the quality of training data for L2R).

In a typical active learning scenario, data instances are selected from an unlabeled set one at a time and labeled by a human expert. Every time a new sample is selected and labeled, a new learning model is produced and the active learning algorithm again chooses a new instance from the unlabeled set. This process is repeated as long as necessary or until the labeling budget is exhausted. After enough instances have been selected and labeled, we can use the selected set as a training set for a learning to rank system.

Figure 3.1 shows the relationship of an active learning system to the other compo-

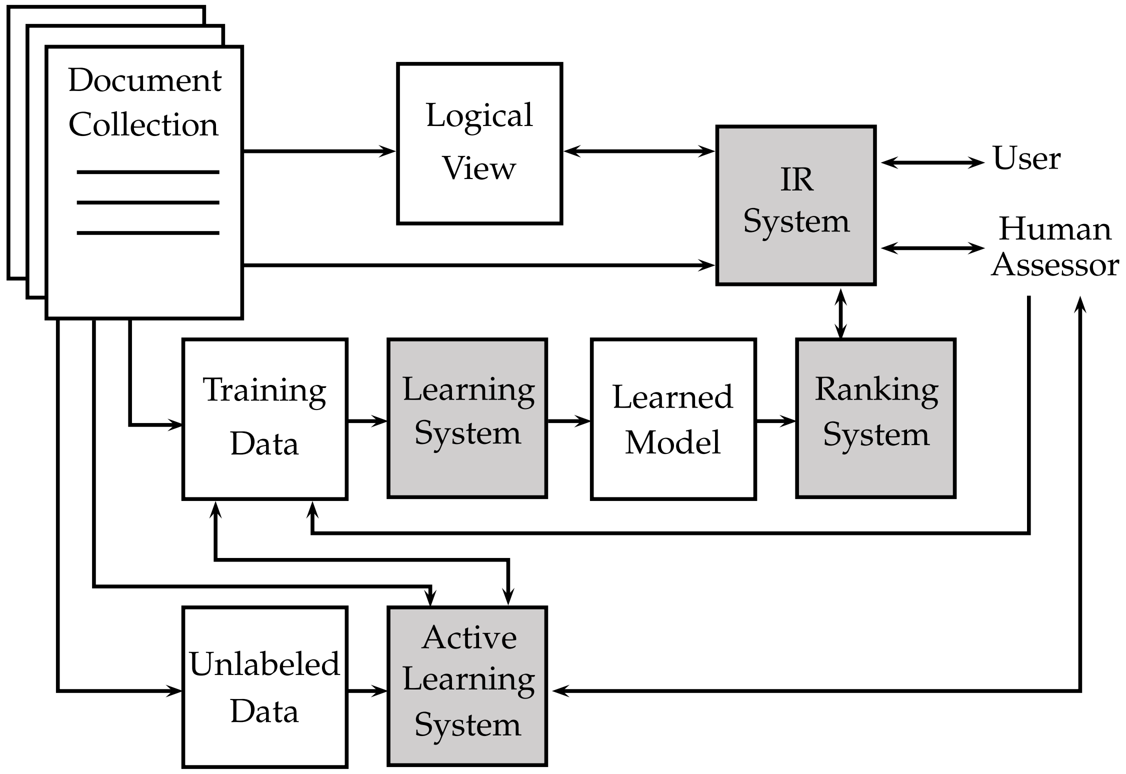

Figure 3.1. Active Learning Framework

nents of an IR/L2R framework. Similarly to the L2R framework described in the last chapter, we first pose a series of sample queries to the IR system, obtaining the lists of documents associated to these queries. We then process these documents, producing an unlabeled set containing the feature-based representation of the query-document instances. The Active Learning System selects documents from this unlabeled set, and presents them to the Human Assessor for relevance evaluation. These labeled samples are put into the training set which the Active Learning System uses to update it's own learning model (not shown in the picture) and proceed selecting more unlabeled samples for labeling. This is repeated as long as necessary (we will later discuss how we can decide when to stop labeling). Once we have enough training data, we can use it to train the L2R System, producing a model that can later be used by the Ranking System to rank the results for a user query, as described in the last chapter.

Observe that although in the "typical" scenario the active learning system selects only one sample per round, it is also possible to select instances in batches and then label them all at once, updating the model and repeating the process. Guo and Schuurmans [2008], for instance, propose a batch-oriented active learning method for classification. The method proposed by Donmez and Carbonell [2008] (which we use as a baseline and will discuss at length in Section 4.2 and in Chapter 5) also selects several documents at each round as does the method we propose in Chapter 5.

3 Active Learning Strategies

The purpose of an active learning method is to select instances that will be very "informative" to the supervised learning algorithm that will later use the selected training set.

Several works propose active learning methods for classification tasks (see, for instance, Donmez et al. [2007]; Mccallum [1998]; Nguyen and Smeulders [2004]; Schohn and Cohn [2000]; Tong and Koller [2002]). While classification functions output a distinct class for each data item, ranking functions must produce partial orders of items either through some scoring function, pairwise ordering or listwise ordering of the items. Most active sampling methods for classification try to directly minimize the classification error, but this approach is not straightforward to extend to the ranking problem since, as noted by Liu [2009], position-based measures such as MAP and NDCG are usually non-continuous and non-differentiable. Additionally, in most supervised learning settings, samples can be treated as independent of each other which is not the case for L2R where each sample represents a document *relative to a query*. Thus, in L2R, instances are conditionally independent given a query (see Long et al. [2010]).

Despite the differences between classification and L2R, active learning methods proposed in both fields have common general outlines or strategies. In some methods, the most ambiguous, or those instances for which the learner is most *uncertain* about, are selected (see, for instance, Lewis and Gale [1994]; Tian and Lease [2011]). This approach is simple and straightforward, specially for probabilistic learners in classification. By obtaining the label of a sample lying close to the boundary between one class and another the classifier can better discriminate instances of each class. For a binary probabilistic classifier, this approach amounts to choosing the instances whose posterior probability of being positive is nearest 0.5. For multiple class problems, *margin sampling* (see Scheffer et al. [2001]) or *entropy* (see Settles and Craven [2008]) based strategies may be used.

In other settings, a query-by-committee (QBC) strategy is employed where competing learners vote on the label of the candidate samples and those about which they most disagree are chosen (Cai et al. [2011]; Seung et al. [1992]). The idea behind QBC-based active methods is that of minimizing the version space (or the set of hypotheses consistent with the current labeled training data), thus making the search for a good hypothesis easier. See Chapter 5 for more information on QBC strategies.

Some algorithms select the samples that would cause the greatest change to the currently learned function (for example, Donmez and Carbonell [2008]; Settles et al. [2008]). This is the case of the active method for L2R proposed by Donmez and Carbonell [2008] which we detail in Section 4.2. The intuition behind this strategy is that in an active learning scenario, we usually start with a very simple and incomplete learning model and that by incorporating into the training set those samples that (we estimate) will change the model the most, we can accelerate the model's evolution towards an effective hypothesis.

Other methods select instances that would lead to the minimal expected future error or, similarly, optimize some other metric such as precision or recall (for instance, Donmez and Carbonell [2009]; Settles [2009]). These approaches can be computationally expensive as they usually involve not only the future error estimation, but also the retraining of the learned model for each possible label of the instances considered (although, in some cases, it is possible to incrementally update the model, this is also possibly expensive). Some other approaches, such as Long et al. [2010], minimize the loss function indirectly by minimizing the output variance.

4 Related Work

4.1 Active Learning for Learning to Rank

Some researchers have recently proposed active learning schemes for L2R based on the optimization of approximations of position-based measures. The authors of Long et al. [2010], for example, propose a general active learning framework based on expected loss optimization (ELO). The proposed framework uses function ensembles to select training examples that minimize a chosen loss function. The authors approach the unique challenges of active learning in L2R by separating their selection algorithm into query level and document level parts (what they call *Two-stage Active Learning*). To approximate their chosen metric, namely, DCG (*Discounted Cumulative Gain*), they use ensembles of learners to produce relevance scores and estimate predictive distributions for the documents in the active learning set. To produce the ensemble, they use a bootstrap technique that relies on an initial labeled set. Thus, their technique requires a initial labeled set to build the ensemble of learners, which needs to be large enough for the learners in the ensemble to be minimally effective. They test their method using a large commercial Web search dataset (500K documents) and using initial (labeled) sets of 2K, 4K and 8K documents.

An SVM-specific strategy is presented by Donmez and Carbonell [2009]. The authors rely on the relationship between the area under the ROC curve (AUC) and the hinge rank loss proposed by Steck [2007] to develop a loss minimization framework for active learning in ranking. Instead of testing each and every unlabeled sample to determine the one that has the smallest expected future error, the authors suggest selecting the examples that have the largest contribution to the estimated current error. These are potentially the ones that will bring more benefit when labeled for the functions that will be trained in the next rounds of the method. The proposed selection criterion is based on the hinge rank loss calculated on a per query basis and depends on the determination of a rank threshold that estimates the rank position that separates the lowest ranked relevant element from the highest ranked non-relevant example. The algorithm starts with a small labeled per query set and proceeds selecting unlabeled samples that have the highest uncertainty (as defined by the rank threshold). These samples are then labeled and added to the per query labeled sets and the process is repeated as many times as desired.

A slightly different approach is proposed by Donmez and Carbonell [2008]. Their method relies on the estimated risk of the ranking function on the labeled set after adding a new instance with all possible labels. The authors present results using this sampling technique with SVMRank and RankBoost. Their method, which also relies on an initial labeled training set and incrementally adds new samples, achieves competitive results with around 15% of the original training sets. We have chosen to implement this method as a baseline because it is elegant and simple and provides very good results. We describe this method in more detail in the next section.

Some other works apply active learning strategies in association with other techniques such as transfer learning or relevance feedback. Cai et al. [2011], for instance, propose a method integrating Domain Adaptation and active learning as a way to reduce labeling costs. They first use a QBC scheme built on a mixture of source-domain and target domain data to select the most "informative" queries from the target domain. Then these new labeled sets are used to adjust the importance weights of source-domain queries for boosting the transfer of prior ranking knowledge. Their QBC strategy is based on the query-by-bagging concept by Abe and Mamitsuka [1998] where from the currently labeled set a number of partitions is created by sampling uniformly with replacement and the same learning algorithm is trained using these partitions to obtain different models to be used as the committee. Their method performs only query-level selection, meaning that all the documents from the selected queries have to be labeled. Although their method is not directly comparable to ours (since it uses rank adaptation on top of active learning), it must be noted that by using query-level selection only, the amount of labeled documents grows very fast for datasets such as those in LETOR 3.0 (which are used to evaluate their method). In contrast, our work uses an ensemble of learners approach to QBC and does document-level selection (although, as we discuss later, it can be easily extended to use query, document or query-document level selection), achieving better results using 15% (or less) of the unlabeled sets than those presented in their work using 100% of the training sets.

Another work leverages active learning using Relevance Feedback (RF). Tian and Lease [2011] propose a method that iteractively improves the rankings shown to the user based on the feedback of which link the user clicks at each round. The method uses a Support Vector Machine (SVM) algorithm to classify the documents into relevant and not relevant. Their active learning scheme uses two approaches: in the Simple Margin version, those documents lying closest to the decision hyperplane are considered the ones the SVM is most *uncertain* about. They also propose a variation, called Local Structure, which also takes into account the proximity of the unlabeled instances to already labeled data. This variation chooses instances close to the decision surface but also far from already labeled data and close to more unlabeled samples in an attempt to maximize the *diversity* of the selected set and improve the algorithm's learning curve. They evaluate their method using Robust04 and LETOR 3.0, comparing it to RF baselines. Their work is not directly comparable to ours, since we aim at providing a method for a training set creation effort where the human annotators evaluate each document selected by the system in turn, doing that until the (small) labeling budget is met.

4.2 The Donmez Method

Here we describe the active learning for rank method proposed by Donmez and Carbonell [2008]. Henceforth we refer to this method as "Donmez".

The method starts with a per query labeled seed set. It progresses in rounds, selecting a preset number of new documents per query that are then labeled and added to the training set. The basic idea is to estimate, for each query, what is the capacity of an unlabeled example to update the current model if it is labeled and added to the training set. The top k samples (for each query) that have the highest estimated impact in the current model are selected and labeled. We will see in detail the practical aspects of the Donmez method in Chapter 5, where we use it as a baseline to compare our own round-based method against. Here we briefly discuss the theoretical framework of the method.

The basic idea of Donmez's method is to estimate the impact that each unlabeled sample would have in the current SVMRank model without having to retrain the model including each unlabeled sample with all the possible labels (which would be computationally expensive). This general framework is applied to SVMRank in the following manner.

Given a family of ranking functions F, we want to find a linear function $f \in F$ of the form $f(\vec{x}) = \langle \vec{w}, \vec{x} \rangle$ that satisfies $\vec{x}_i \succ \vec{x}_j \Leftrightarrow \langle \vec{w}, \vec{x}_i \rangle > \langle \vec{w}, \vec{x}_j \rangle$ where $\vec{w}$ is the weight vector adjusted by learning and $\vec{x}$ is the feature vector of a query-document instance.

The soft-margin SVM model proposed by Vapnik [1995] is used by Joachims [2002] to target the problem of finding f formulating it as a quadratic optimization problem:

$$\min_{\vec{w}} \frac{1}{2} \parallel \vec{w} \parallel^2 + C \sum \xi_{ij} \tag{3.1}$$

$$\text{subject to } \langle \vec{w}, \vec{x}_i \rangle \geq \langle \vec{w}, \vec{x}_j \rangle + 1 - \xi_{ij},\ \ \xi_{ij} \geq 0\ \forall i, j$$

where C is a parameter that allows trading off margin size for training error. Donmez and Carbonell rewrite this optimization in terms of the hinge loss, substituting C for $\lambda = \frac{1}{2C}$ and obtaining

$$\min_{\vec{w}} \sum_{k=1}^{K} [1 - z_k \langle \vec{w}, \vec{x}_k^1 - \vec{x}_k^2 \rangle]_+ + \lambda \parallel \vec{w} \parallel^2 \tag{3.2}$$

where K is the total number of pairwise comparisons, $\vec{x}_k^1 - \vec{x}_k^2$ is a pairwise difference vector whose label z is positive (i.e. $z = 1$) if $\vec{x}_k^1 \succ \vec{x}_k^2$ and negative (i.e. $z = -1$) otherwise and $[1 - z_k \langle \vec{w}, \vec{x}_k^1 - \vec{x}_k^2 \rangle]_+$ is the hinge loss.

Now suppose we were to include a sample $\vec{x}$ from the unlabeled set $\mathcal{U}$ into our training set. The total loss of the instance pairs containing $\vec{x}$ would be given by the function of $\vec{x}$ and $\vec{w}$ $D(\vec{x}, \vec{w}) = \sum_{j=1}^{J} [1 - z_k \langle \vec{w}, \vec{x}_j - \vec{x} \rangle]_+$ where J is the number of examples already in the training set that have a different label than $\vec{x}$ (remember that pairwise comparisons using relevance levels are only useful when comparing documents with different labels). Then the function to be minimized becomes:

$$\min_{\vec{w}} \left\{ \lambda \parallel \vec{w} \parallel^2 + \sum_{k=1}^{K} [1 - z_k \langle \vec{w}, \vec{x}_k^1 - \vec{x}_k^2 \rangle]_+ + D(\vec{x}, \vec{w}) \right\} \tag{3.3}$$

Let's say that $\vec{w}^*$ is the solution to the optimization problem in 3.2 (i.e. the weight vector in our current learned model) and that $\hat{\vec{w}}$ is the vector that minimizes $D(\vec{x}, \vec{w})$. If $\vec{w}^* \neq \hat{\vec{w}}$ then as the difference $\parallel \vec{w}^* - \hat{\vec{w}} \parallel$ increases it becomes less likely that $\vec{w}^*$ is optimal for Equation 3.3. In other words, the higher this difference, the more $\vec{w}^*$ needs to be updated in order to compensate for the loss on the new pairs.

Let's write $\hat{\vec{w}}$ in terms of $\vec{w}^*$ as $\hat{\vec{w}} = \vec{w}^* - \Delta w$. To obtain $\hat{\vec{w}}$, we need to minimize the objective function:

$$\min_{\vec{w}} D(\vec{w}, \vec{x}) = \min_{\vec{w}} \sum_{j=1}^{J} [1 - z_j \langle \vec{w}, \vec{x_j} - \vec{x} \rangle]_+ \tag{3.4}$$

The derivative of this equation with respect to a single point $\vec{x_j}$, $\Delta\vec{w_j}$ is:

$$\Delta\vec{w_j} = \begin{cases} 0 & \text{if } z_j \langle \vec{w}, \vec{x_j} - \vec{x} \rangle \geq 1 \\ -z_j(\vec{x_j} - \vec{x}) & \text{if } z_j \langle \vec{w}, \vec{x_j} - \vec{x} \rangle < 1 \end{cases} \tag{3.5}$$

Substituting $\vec{w}$ in Equation 3.5 for the current weight vector $\vec{w}^*$, we can estimate how the solution of Equation 3.4 deviates from it. That is, $\parallel \vec{w}^* - \hat{\vec{w}} \parallel = \parallel \Delta\vec{w} \parallel$. Then we can write the magnitude of the total derivative as a function of $\vec{x}$ and the rank label y as:

$$\begin{aligned} g(\vec{x}, y) &= \parallel \Delta\vec{w} \parallel = \textstyle\sum_j \parallel \Delta\vec{w_j} \parallel \\ &= \textstyle\sum_{j=1}^{J} \begin{cases} 0 & \text{if } z_j \langle \vec{w}^*, \vec{x_j} - \vec{x} \rangle \geq 1 \\ \parallel -z_j(\vec{x_j} - \vec{x}) \parallel & \text{if } z_j \langle \vec{w}^*, \vec{x_j} - \vec{x} \rangle < 1 \end{cases} \end{aligned} \tag{3.6}$$

Thus, $g(\vec{x}, y)$ estimates how the current model will change with the addition of the document $\vec{x}$ with label y. Remember that we are evaluating the documents in the unlabeled set and, therefore, we do not know the true label y of each instance $\vec{x}$. But we can use the current model to estimate the label probabilities $\hat{P}(y|\vec{x})$ for all y. In a binary relevance scenario (i.e. $y \in \{0, 1\}$) we evaluate all unlabeled documents for each query in turn and choose $\vec{x}^*$ according to:

$$\vec{x}^* = \underset{\vec{x} \in \mathcal{U}}{\operatorname{argmax}} \left\{ \hat{P}(y = 1|\vec{x}) \times g(\vec{x}, y = 1) + \hat{P}(y = 0|\vec{x}) \times g(\vec{x}, y = 0) \right\} \tag{3.7}$$

As we will see in more detail in Chapter 5, Donmez's method actually selects the top 5 documents for each query at each round. Therefore, in each round we sort all documents $\vec{x}$ of a query q in descending order of the value obtained using Equation 3.7 and select the top 5 documents to be labeled and added to the training set.

Notice that one of Donmez's main assumptions is that the documents expected to change the current model the most will lead to learning the true hypothesis faster. It is possible that in some rounds the documents selected lead to worse generalization of the model (i.e. the selected documents are "noisy" to the learning algorithm), but as more rounds are run (and more documents labeled and added to the training set) their influence will be mitigated as the labeled set increases in size.

By using the probabilities of the labels $\hat{P}(y|\vec{x})$ that the current model predicts for each instance in the estimation of $\vec{x}^*$, Donmez's method also incorporates an element of *uncertainty*-based sampling. This is one of the most common active learning strategies for learners that output class probabilities. Samples that have probabilities closer to 0.5 may be chosen more often if the estimated model correction $g(\vec{x}, y)$ is high for either class or for both. On the other hand, samples where one of the projected classes has a higher probability and when the estimated model change for the same class is also high may also be chosen more frequently.

By mingling estimated model change and uncertainty sampling, Donmez method is very elegant and effective as we will see in Chapter 4 and in Chapter 5.

5 Summary

Active learning methods provide a mechanism to actively select "informative" unlabeled samples from a pool of instances so as to build an effective training set for supervised learning algorithms. By allowing the active learning algorithm to select the samples that have to be labeled, we hope to reduce the considerable cost of labeling documents. Observe that it is inexpensive to obtain unlabeled samples: all that needs to be done is to use sample queries (gathered from query logs, for instance) to obtain documents using the IR system and then generate the query-document features used by the active learning algorithm. The active learning method then selects instances from this pool of unlabeled examples and present them to the human assessor to be labeled.

In this chapter, we have briefly described some of the most common active learning strategies used in classification and ranking. We also presented some of the active learning methods that have been proposed for ranking in the last few years, describing in detail one such method (i.e. "Donmez") which we will use as a baseline to evaluate the novel methods we propose in the next two chapters.

Chapter 4

Active Learning to Rank Using Association Rules

1 Introduction

In this chapter we propose a new active learning technique based on association rules. Learning to rank using demand-driven association rules has been shown to provide competitive ranking quality by Veloso et al. [2008]. The method proposed here uses association rules to actively select documents from an unlabeled set based on how many inference rules are generated for each document. At each step, a new document is chosen for labeling as the most "dissimilar" with respect to the already selected samples (but with no need to create a model), with the goal of increasing diversity and representativeness. This is performed by choosing from the unlabeled data the document u_i that generates the smallest number of rules when considering a "projection" of the current selected training set with respect to the features of u_i. This projection aims at removing examples from the current training set that are not useful in producing rules for u_i. The more dissimilar the candidate document, the fewer the rules generated for it, meaning that few already labeled instances in the projected training set share common feature-values with the candidate. This process is repeated until the algorithm converges, which happens when an already selected document is selected again, i.e., there is no more information useful for generating rules from any other document in the unlabeled set.

Despite being very effective in several cases, as demonstrated in our experiments, our method may sometimes converge too fast, resulting in a very small labeled training set. We propose a strategy to delay this convergence and increase the size and diversity of the labeled set which consists in vertically partitioning the features in the unlabeled set, generating in practice several reduced (in terms of features, not instances) unlabeled sets, over which our active sampling method can be applied. Once the final reduced training set is created by the union of the documents selected in each partition, we apply the supervised on-demand association rule algorithm (RLR) to rank the test set. One of the key advantages of our method is that it can be directly applied on an unlabeled set containing the extracted features for the documents in the corpus, without the need for an initial labeled set. Furthermore, once the unlabeled set is generated (i.e. feature extraction is performed on the corpus) and processed (i.e. discretized and partitioned), the method can be directly applied without extra parametrization since it naturally converges while selecting the training samples. This is different from most previous work where there is no clear stop criterion and the number of iterations has to be empirically determined.

We compare our method against a number of baselines that use the complete labeled training set, including some published by the LETOR dataset producers with many supervised L2R algorithms, and show that it is competitive when compared to several of them while selecting and using as little as 1.12% and at most 2.28% of the original training sets (average of 1.63% for all datasets considered). Best results reported in the literature with other active sampling strategies for L2R reported selecting at least 11% of the training set to produce similar competitive results. Moreover, in most cases our method improved the results of the original on-demand associative method (by as much as 13%), or produced similar results when compared to using the whole training set, confirming the hypothesis that active learning methods may be able to improve the results by reducing the “noise” in the training sets.

2 Active Learning using Association Rules: ARLR

In this section we present a novel algorithm referred to as ARLR (Active Rule-based Learning to Rank), which relies on an effective selective sampling strategy in order to deal with the high cost of labeling large amounts of examples. The key idea of ARLR is that it may provide results as effective (or even better) as RLR by carefully choosing a much smaller set of training examples from which it learns the ranking functions. ARLR takes each unlabeled document in turn, obtaining its projection from the current reduced training set and generating the rules that would be used to rank the document. After it has the rules for all unlabeled documents, it chooses the one that generated the fewest rules, obtaining its label and inserting it into the current selected and labeled training set. The goal is to label as few instances as possible, while providing equal or even improved ranking performance.

Sampling Function

Consider a large set of unlabeled documents $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$. The problem we investigate in this section is how to select a small subset of documents in $\mathcal{U}$, such that the selected documents carry almost the same information of all documents in $\mathcal{U}$. These highly informative documents will compose the training data $\mathcal{D}$, and, ideally, $|\mathcal{D}| \ll |\mathcal{U}|$. Particularly, ARLR exploits the redundancy in feature-space that exists between different documents in $\mathcal{U}$. That is, many documents in $\mathcal{U}$ may share some of their feature-values, and ARLR uses this fact to perform an effective selective sampling strategy.

Remember from Chapter 2 that RLR ranks a document d of a new query by creating a projection $\mathcal{D}_d$ of the training set in order to extract a set of association rules $\mathcal{R}_d$ used to estimate d's relevance. ARLR takes a document u_i from the unlabeled set and also creates a projection of its small labeled set $\mathcal{D}$ in order to generate rules for ranking. But instead of doing this to estimate u_i's relevance, it is interested in *how many rules are generated.* Intuitively, if a document $u_i \in \mathcal{U}$ is inserted into $\mathcal{D}$, then the number of useful rules for documents in $\mathcal{U}$ that share feature-values with u_i will possibly increase. In contrast, the number of useful rules for those documents in $\mathcal{U}$ that do not share any feature-value with u_i will clearly remain unchanged. Therefore, the number of rules extracted for each document in $\mathcal{U}$ can be used as an approximation of the amount of redundant information between documents already in $\mathcal{D}$ and documents in $\mathcal{U}$. The sampling function employed by ARLR exploits this key idea, by selecting documents that contribute primarily with non-redundant information, and these informative documents are those likely to demand the fewer number of rules from $\mathcal{D}$. More specifically, the sampling function $\gamma(\mathcal{U})$ returns a document in $\mathcal{U}$ according to Equation 4.1:

$$\gamma(\mathcal{U}) = \{u_i \text{ such that } \forall u_j : \; |\mathcal{R}_{u_i}| \; \leq \; |\mathcal{R}_{u_j}|\} \tag{4.1}$$

The document returned by the sampling function is inserted into $\mathcal{D}$, but it also remains in $\mathcal{U}$. In the next round of ARLR, the sampling function is executed again, but the number of rules extracted from $\mathcal{D}$ for each document in $\mathcal{U}$ is likely to change due to the document recently inserted into $\mathcal{D}$. The intuition behind choosing the document which demands the fewest rules is that such document should share less feature-values with documents that were already inserted into $\mathcal{D}$. That is, if only few rules are extracted for a document u_i, then this is evidence that $\mathcal{D}$ does not contain documents that are similar to u_i, and, thus, the information provided by document u_i is not redundant and u_i is a highly informative document. This simple heuristic works in a fine-grained level of feature-values trying to maximize the diversity in the training set. The extracted rules capture the co-occurrence of feature-values, helping in our goal of increasing diversity, since in this case, the document which demands the fewest rules is exactly the one which shares the least possible number of feature-values with documents already in the training data. In the case of a tie, the algorithm selects the document based on the size of the projection.

Notice that initially $\mathcal{D}$ is empty, and thus ARLR cannot extract any rules from $\mathcal{D}$. The first document to be labeled and inserted into $\mathcal{D}$ is selected from the set of available documents $\mathcal{U}$. In order to maximize the initial coverage of $\mathcal{D}$, the selected document is the one that maximizes the size of the projected data in $\mathcal{U}$, that is, it is the document d for which $\mathcal{U}_d$ is the largest. This is the document that shares more feature-values with the other documents of the collection and can be considered as the best representative of it. After the first document is selected and labeled, the algorithm proceeds using the fewest rules heuristic, as described above.

Natural Stop Condition

After selecting the first document and at each posterior round, ARLR executes the sampling function and a new example is inserted into $\mathcal{D}$. At iteration i, the selected document is denoted as $\gamma_i(\mathcal{U})$, and it is likely to be as dissimilar as possible from the documents already in $\mathcal{D}$=$\{\gamma_{i-1}(\mathcal{U}), \gamma_{i-2}(\mathcal{U}), \ldots, \gamma_1(\mathcal{U})\}$. The algorithm keeps inserting documents into the training data, until the stop criterion is achieved.

Lemma 2: If $\gamma_i(\mathcal{U}) \in \mathcal{D}$ then $\gamma_i(\mathcal{U})$=$\gamma_j(\mathcal{U})$ $\forall j>i$.

Proof: If $\gamma_i(\mathcal{U}) \in \mathcal{D}$ then the inclusion of $\gamma_i(\mathcal{U})$ does not change $\mathcal{D}$. As a result, any further execution of the sampling function must return the same document returned by $\gamma_i(\mathcal{U})$, and $\mathcal{D}$ will never change. ■

The algorithm stops when all available documents in $\mathcal{U}$ are less informative than any document already inserted into $\mathcal{D}$. This occurs exactly when ARLR selects a document which is already in $\mathcal{D}$. According to Lemma 2, when this condition is reached, ARLR will keep selecting the same document over and over again. At this point, the training data $\mathcal{D}$ contains the most informative documents, and RLR can be applied to estimate the relevance of documents in $\mathcal{T}$. All steps of ARLR are shown in Algorithm 2.

Algorithm 2 Active Rule-based Learning to Rank: ARLR

Require: Unlabeled data $\mathcal{U}$, and σ_{min} (≈ 0)
Ensure: The training data $\mathcal{D}$

```
 1: repeat
 2:     for all document $u_i \in \mathcal{U}$ do
 3:         $\mathcal{D}_{u_i} \Leftarrow \mathcal{D}$ projected according to $u_i$
 4:         $\mathcal{R}_{u_i} \Leftarrow$ rules extracted from $\mathcal{D}_{u_i} \mid \sigma \geq \sigma_{min}$
 5:     end for
 6:     if $\mathcal{D} = \emptyset$ then
 7:         $\gamma_i(\mathcal{U}) \Leftarrow u_i$ such that $\forall u_j : |\mathcal{U}_{u_i}| \geq |\mathcal{U}_{u_j}|\}$
 8:     else
 9:         $\gamma_i(\mathcal{U}) \Leftarrow u_i$ such that $\forall u_j : |\mathcal{R}_{u_i}| \leq |\mathcal{R}_{u_j}|\}$
10:     end if
11:     if $\gamma_i(\mathcal{U}) \in \mathcal{D}$ then break
12:     else append $\gamma_i(\mathcal{U})$ to $\mathcal{D}$
```

3 Experimental Evaluation

3.1 Experimental Setup

As described in Section 3, ARLR, our rule-based active sampling algorithm processes a given list of unlabeled instances selecting the one that produces the fewest rules. In this setup, the training set for the selection process is initially empty and grows one instance at a time as they are selected from the unlabeled set and labeled. The algorithm eventually converges when it selects an instance it has already selected before. Once that happens, the selected items can be used as a reduced training set to rank the test set using RLR (i.e. the supervised rule-based rank-learner). All results presented here use, therefore, two distinct sets. The original training set is used as the unlabeled set from which instances are selected by ARLR. The test set is then ranked by the RLR using the selected and labeled instances as training. Observe that our method does not use an initial labeled set to learn an initial model. The labeling effort is restricted to the examples selected by the algorithm directly from the unlabeled set. In our experiments, the presence of the user who would provide the labels is simulated; we use instead the original labels available in the collection after a document is selected.

The association rule mining algorithm uses nominal values to generate the inference rules used in ranking the results or in selecting samples. Therefore it is necessary to discretize the original LETOR data. Since all our data is unlabeled, we need to use an unsupervised discretization algorithm. Simple algorithms such as Uniform Range Discretization (URD) or Uniform Frequency Discretization (UFD) could be used, but being oversimplistic, they may cause some loss of information. Instead, we use the

Table 4.1. ARLR without Partitioning MAP Results and Statistics

	ARLR	RLR	BM25	Random	G%	Sel#	Sel%	RS%	R%	R25%
TD2003	0.2032	0.2459	0.1402	0.1181±0.0234	44.94	157	0.53	6.80	1.52	13.22
TD2004	0.1792	0.2463	0.1452	0.1267±0.0163	23.47	141	0.32	7.82	2.92	11.32
NP2003	0.7202	0.6373	0.5346	0.4981±0.0688	34.72	207	0.23	3.04	0.10	25.32
NP2004	0.4993	0.5155	0.2672	0.3695±0.0575	35.15	181	0.41	3.18	0.17	5.76
HP2003	0.6487	0.7083	0.5230	0.5486±0.0493	18.25	218	0.25	3.62	0.12	26.04
HP2004	0.6332	0.5443	0.3712	0.3117±0.0496	70.55	222	0.50	1.68	0.13	4.24

Tree-based Unsupervised Bin Estimator (TUBE) proposed by Schmidberger and Frank [2005]. TUBE is a greedy non-parametric density estimation algorithm that uses the log-likelihood measure and a top-down tree building strategy to define varying-length bins for each attribute in the dataset. The algorithm can automatically determine the number of bins using cross-validation and the log-likelihood. Since this approach can lead to too many bins in some cases, we chose to use 10 varying-length bins for all attributes in all datasets (see Section 3.3.1 for an analysis of the discretization process). The results shown in tables 4.1 and 4.2 were obtained using TUBE to discretize each feature from the training set into 10 bins. After the bins were determined using the training sets in each fold, the test sets were discretized using the same bins. All results presented use 5-fold cross-validation on the query-normal version of the datasets.

3.2 Results

Table 4.1 presents the results for this initial setup. The first column, "ARLR", shows the MAP obtained using only the samples selected from the unlabeled set. The number of selected samples appears in the "Sel#" column (for the total number of instances in the unlabeled set, see the "TR Size" column of Table 2.1). The "Sel%" column indicates the percentage of the unlabeled set that the selected examples represent. The "RS%" column shows the proportion of relevant samples in the selected set while the "R%" field shows the proportion of relevant documents in the full training set. "R25%" shows the proportion of relevant samples selected by the BM25 baseline described below. As a reference result, the "RLR" column contains the MAP obtained by the RLR algorithm using the complete training set and supervised discretization. We show this result for comparison, since it uses the very effective supervised discretization method proposed by Fayyad and Irani [1993] and provides a target to measure our hypothesis of noise reduction. The baselines appear on the 3^{rd} and 4^{th} columns: "BM25" shows the resulting MAP from running RLR with the same amount of samples selected by ARLR as training but selected using the value of the BM25 attribute in descending order (i.e.

instances with the highest BM25 were used); "Random" shows the MAP obtained by randomly selecting the same amount of samples selected by ARLR and using these as the training set for supervised RLR ranking. Finally, the "G%" column indicates the gain obtained by ARLR over the best value from the BM25 and Random baselines. The Random baseline was produced by randomly sampling the same amount of instances selected by ARLR at least 20 times for each fold and averaging the resulting MAPs. We present the mean obtained from all runs and all folds as well as the confidence interval for a confidence level of 95%. The RLR with BM25 baseline tries to select more interesting instances by using the BM25 attribute value as a measure of instance quality.

From Table 4.1 we can see that the method converges very fast, selecting from 0.23 to 0.53% of the instances in the unlabeled set. Compared to the baselines, it performs very well as can be seen on the "G%" column. ARLR even beats the RLR reference result in NP2003 and HP2004. Notice also that ARLR tends to select a much higher proportion of relevant instances than present in the original sets (RS% vs. R%). These datasets have an extremely reduced amount of relevant instances and ARLR seems to single them out based on the "fewer rules" heuristic. On the other hand, the BM25-based selection obtains an even higher proportion of relevant documents, showing the power of this classic IR measure. But from the BM25 results we can see that it is not only a matter of using many relevant instances, but also about the overall "quality" of the instances used. Our rule-based selection method does choose many relevant instances, but it does so based on the rules created from all attributes.

These initial results are promising, but the algorithm is converging too fast in some collections for the selected samples to have sufficient representativeness and diversity. Next we propose a simple and yet effective strategy to delay the convergence.

3.2.1 Increasing Sample Diversity

The approach described above has two potential issues: first, it may take some time to run on datasets with too many features, since the active sampling algorithm's complexity depends on the number of features and the size of the unlabeled set. Second, as we can see in Table 4.1, the number of instances selected using this method is very small, ranging from 0.23 to 0.53% of the unlabeled set size. There's no doubt that the selected samples are very informative, since the results obtained by ARLR are usually reasonably better than the baselines. Nonetheless, the resulting training dataset may still be too small to provide, in some collections, the minimum diversity of examples for the supervised algorithm to reach a performance comparable to that obtained using the complete training set (i.e. column “RLR” in Table 4.1). By delaying convergence and increasing the number of sampled instances we can perhaps improve the diversity of the selected set.

We propose partitioning the unlabeled set into vertical sections containing subsets of the features[1]. Each of these partitions contain all unlabeled instances, but only a group of each instance's features. The active sampling algorithm can then be run on each partition, selecting instances based on distinct feature sets. This simple strategy increases the number of selected instances, since for each partition the algorithm will choose those that are most informative given the features in that partition. It may also improve the diversity of the samples, as they are selected based on distinct characteristics. To apply this strategy, we need to determine how to separate the features into the partitions and how many partitions should be used. The features need to be divided in such a way as to allow the algorithm to select informative samples regardless of which feature set is used. However, features have diverse informational values, with some being more informative to the rank learning algorithm than others. Ideally, we want to distribute the most informative features through all partitions in order to maximize the quality and variety of the instances selected by ARLR from each partition.

There are many ways to estimate which features provide more information. We chose to estimate the χ^2 value of each feature. Since we do not have relevance information, we cannot calculate the χ^2 in relation to the label. What we do instead is to calculate the χ^2 of each feature in relation to the others (i.e. how well one feature performs in predicting another feature's value). Thus, for m features, we produce a matrix containing in line i the $m-1$ features ranked in descending order of their χ^2 value in relation to the i_{th} feature. We then combine the m rankings by scoring each feature according to its positions in all rankings. If a feature appears at the first position in a ranking we add 1 to its score, otherwise, we add $1/log(10 \times j)$ where j is the feature's position in that ranking. Then we order the features in descending order of score, obtaining the final ranking.

With this ranked list of features, we can now vertically partition the unlabeled set by spreading the features into the partitions in the ranked order. Thus, given the ranked list of m features $\mathcal{F} = \{f_1, f_2, ..., f_m\}$, and a set of n partitions $\mathcal{P} = \{\mathcal{U}_1, \mathcal{U}_2, ..., \mathcal{U}_n\}$, we place feature f_1 (the one in the first position of the χ^2 ranking) in partition $\mathcal{U}_1$ then f_2 in $\mathcal{U}_2$, eventually reaching partition $\mathcal{U}_n$ and starting over by placing the next feature, f_i into $\mathcal{U}_1$ and f_{i+1} into $\mathcal{U}_2$ and so on. We now run ARLR using each $\mathcal{U}_k$ set as input, and obtaining n sets of selected documents $\mathcal{D}_k$. We then obtain the original set of features

[1]Another strategy would be to select documents *per query*, but this would entail a large number of actively selected samples, thus hurting our goal of significantly reducing the labelling effort.

Algorithm 3 ARLR with Partitioning

```
Require: Unlabeled data $\mathcal{U}$, Ranked list of features $\mathcal{F}$, Number of partitions $n = |\mathcal{P}|$
Ensure: The training data $\mathcal{D}$
 1: for all $i \mid 0 \leq i < |\mathcal{F}|$ do
 2:     $j \Leftarrow i$ mod $n$
 3:     $\mathcal{U}_{j+1} \Leftarrow$ projection of $\mathcal{U}$ with all values for $f_{i+1}$
 4: end for
 5: for all $k \mid 1 \leq k \leq |\mathcal{P}|$ do
 6:     $\mathcal{D}_k \Leftarrow$ ARLR($\mathcal{U}_k$, $\sigma_{min}$)
 7: end for
 8: for all $k \mid 1 \leq k \leq |\mathcal{P}|$ do
 9:     $\mathcal{D} \Leftarrow \mathcal{D} \cup \{d_j \in \mathcal{U} | \exists d_i \in D_k, i = j\}$
10: end for
```

for all selected documents from $\mathcal{U}$ and run RLR using this new set as training. A brief description of this version of ARLR with Partitions is shown in Algorithm 3.

The next step is to determine how many partitions to use. If very few partitions are used, then the increase in document diversity and the number of selected instances may be small, yielding only marginal gains. If we use too many partitions, then the informational value of each document is diluted and ARLR will select many instances that are not informative as a whole, but only when considered in the context of the reduced feature set (i.e. the partition from which it was selected). Thus it is important to determine how many partitions to use to obtain good diversity while selecting informative samples. We performed preliminary tests using 2 collections (see Section 3.3.2 for a more detailed analysis of the effect of partitioning on the results), and decided to set the number of features per partition to be from around 8 to 12. Using 5 partitions for all datasets, we have around 12 features per set for LETOR 3.0. For datasets with more features, more partitions should be used.

Table 4.2 presents the results for ARLR. All results were produced by splitting the unlabeled sets in 5 partitions, selecting instances from the partitions, creating a new labeled training set comprised of all the selected samples with all features and running RLR on the test set with 5-fold cross validation. Again we compare the resulting MAPs with two baselines: BM25 and Random. The baselines were ran again, since the number of documents selected has changed for all datasets. Notice that the results for "RLR" are the same as in Table 4.1, since it uses the complete training set.

As we can see, ARLR with partitions selects from 4 to 5 times the amount of instances selected by ARLR without partitioning although the percentages relative to the unlabeled set (column "Sel%") remain very low. These percentages now vary from 1.12% (NP2003) to 2.28% (TD2003). Not only the size of the selected set increased,

Table 4.2. ARLR with 5 Partitions MAP Results and Statistics

	ARLR	RLR	BM25	Random	G%	Sel#	Sel%	RS%	R%	R25%
TD2003	0.2728	0.2459	0.2104	0.1573±0.0198	27.84	670	2.28	4.40	1.52	7.70
TD2004	0.2185	0.2463	0.1818	0.1707±0.0112	10.37	593	1.33	5.72	2.92	6.00
NP2003	0.6960	0.6373	0.6321	0.5573±0.0423	10.09	995	1.12	2.42	0.10	7.28
NP2004	0.5499	0.5155	0.4064	0.4038±0.0580	35.29	860	1.94	1.80	0.17	2.16
HP2003	0.7411	0.7083	0.6487	0.5825±0.0460	14.25	1091	1.23	2.00	0.12	6.90
HP2004	0.6168	0.5443	0.3685	0.3718±0.0479	65.89	855	1.91	1.68	0.13	1.74

but there may be more diversity in the selected set as instances were chosen based on different feature groups. As a consequence, the MAP for ARLR with partitioning is better for 4 datasets. The improvement over ARLR without partitioning was 32% for TD2003, 22% for TD2004, 14% for HP2003 and 10% for NP2004. For NP2003 and HP2004, there was a small reduction of around 3%. The proportion of relevant instances selected is also lower for both ARLR and the BM25 baseline. Our method is not only still better than the baselines but also beats RLR using the full training sets on all datasets except TD2004. This is an indication that there is some noise reduction in the active selection process. With only 2% of the original training set it is possible to obtain better results than using the full training set with a supervised method such as RLR.

By vertically partitioning the unlabeled set we are able to increase the number of the samples selected by ARLR. From now on, any reference we make to "ARLR" relates to the full method, including 10 bin discretization and 5 partitions (for the LETOR 3.0 datasets). In the following section, we study in more depth the discretization process which greatly influences the effectiveness of the method. Then, in section 3.3.2 we also analyze the partitioning process at length to better understand how it affects ARLR.

3.3 Analysis

3.3.1 Discretization

RLR and ARLR derive rules with nominal feature-values in the antecedent and relevance levels in the consequent. For these rules to be applicable to documents in the test set (i.e. at query time) real-valued features need to be discretized into a finite set of values that correspond to the discrete values used in the training set. Thus, the *discretization* process consists in dividing the full value interval for each feature into bins (or several intervals) so that real numbers can be mapped onto nominal values (i.e. each bin).

When using supervised algorithms such as RLR, the labeled training set can be used to discretize both the training and the test sets using knowledge of the label to determine the bins for each feature in such a way as to provide good label coherence. One of the most well known supervised discretization algorithms, proposed by Fayyad and Irani [1993], uses a class entropy measure to recursively partition a feature's value range into bins. It uses the Minimum Description Length Principle (MDLP) to decide when to stop partitioning a given range. We cannot use a supervised discretization method in our active learning setup, since we are sampling from an unlabeled set and we need the data discretized before we can sample and label the selected instances. Furthermore, the training sets selected by ARLR are too small to provide enough feature-values for a supervised discretization method to be effective *after* we have labeled the selected instances (and before we proceed to use RLR to rank the test set using the selected sets as training). We are, therefore, bound to use an unsupervised discretization method to prepare the data for instance selection by ARLR and, at query time, RLR to obtain the actual ranking.

The discretization method used to produce the results presented in Tables 4.1 and 4.2 was the Tree-based Unsupervised Bin Estimator (TUBE) proposed by Schmidberger and Frank [2005]. TUBE is a non-parametric density estimation method that relies on the log-likelihood and cross-validation to build a density estimation tree in a top-down fashion. For each range, the algorithm finds the locally optimal binary split (i.e. the one that maximizes the likelihood) and repeats the process recursively in all subranges until the stopping criterion is met or a user-specified number of bins is reached. If we allow the algorithm to determine the number of bins, the stopping criterion (i.e. the final number of cuts) is determined by using a 10-fold cross-validation process where the average log-likelihood over the test folds is computed for all trees with 1 up to $N-1$ cut points and the number of splits that yield the maximum value is chosen. The cross-validation process runs first to determine the final number of cuts for a given attribute and then the algorithm runs one more time to determine the final bins for that feature. If we allow TUBE to automatically determine the number of bins using the stopping criterion explained above, most attributes end up having dozens or even hundreds of bins. With so many bins, ARLR will select a very large number of instances, defeating our objective of keeping the selected sets small. This happens because as the number of distinct feature-values increases (e.g. more bins per feature), instances become more distinct from each other (e.g. some attribute values that fell within the same bin will not do so when more bins are used). In other words, as we increase the number of bins used to discretize the features, the instances become more *diverse* and ARLR selects more instances, since it is built to select samples based

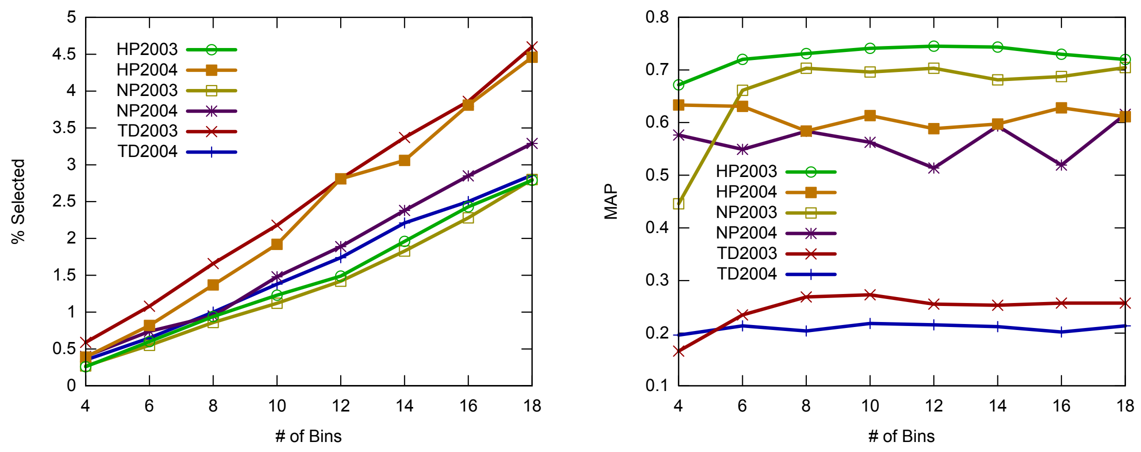

Figure 4.1. % Selected samples vs. # of Bins (left) and MAP vs. # of Bins (right)

on how distinct they are from those already selected. Therefore, in all experiments described in section 3.2 we have used 10 bins to discretize the datasets using TUBE.

We can see the effect of varying the number of bins on the graph at the left of Figure 4.1. It shows the percentage of instances selected from the unlabeled sets as we use more bins to discretize the datasets. Observe that there is an almost linear correlation between the number of bins and the proportion of selected samples. As the number of bins increases, sample diversity also increases and ARLR selects more instances as a consequence.

To the right of Figure 4.1, we can see a plot of the MAP obtained for each dataset as we vary the number of bins (and, consequently, the number of selected samples). For most datasets, the resulting MAP does not vary much as we discretize using more than 8 bins (except for NP2004 and HP2004, where the MAP varies more widely). This indicates that the sets selected by ARLR are almost equally representative of the original unlabeled sets as we increase the number of bins, allowing RLR to effectively rank the test sets. With more bins, there is more variety, and ARLR selects more instances; but this variety and bigger selected sets do not translate into better results, since it is artificially created by the discretization process.

From these results we can see that allowing TUBE to automatically determine the number of bins would be a mistake, since too many instances would be selected without any improvement to the ranking quality. Therefore, we have decided to use 10 bins to discretize all attributes in the datasets in the experiments described in section 3.2 above.

To better understand the quality of TUBE discretization, we ran some tests to

Table 4.3. MAP for RLR using the complete sets with different discretizations

	Fayyad	**TUBE**	**URD**	**UFD**
TD2003	0.2459	**0.2622**	0.2272	0.2179
TD2004	**0.2463**	0.2338	0.1855	0.2306
NP2003	0.6373	0.6343	**0.6788**	0.6328
NP2004	0.5155	**0.5743**	0.5554	0.5226
HP2003	0.7083	0.6998	0.6613	**0.7370**
HP2004	0.5443	0.5519	0.4797	**0.6229**
Avg.	0.4829	0.4927	0.4647	0.4940

compare different discretization methods. We wanted to know how TUBE fares against other unsupervised discretization algorithms and against the widely used supervised method by Fayyad and Irani. Two very simple unsupervised discretization methods are Uniform Range Discretization (URD) and Uniform Frequency Discretization (UFD). In the former, each attribute's range is divided in equal-width bins, where the number k of bins is a user provided parameter. In the latter, the k bins are defined so that each variable-sized bin contains approximately the same number of data points. To be able to evaluate the quality of TUBE independently of ARLR, we ran a supervised ranking test using these discretization algorithms. That is, we discretized each dataset using these four methods and then used RLR to rank the test sets using the complete training sets. For the unsupervised methods, the training set is discretized first and then the test set is discretized using the same bins defined for the training set. The supervised Fayyad method automatically detects the number of bins for each attribute. For the unsupervised methods, we have used 10 bins for all attributes and datasets.

Table 4.3 shows the MAP obtained using RLR on the complete datasets using each discretization technique in turn. Results in **bold** indicate the best result for that dataset. As we can see, TUBE fares very well, beating all other methods in two datasets (TD2003 and NP2004). Compared to Fayyad, TUBE is also better on HP2004 and comes very close in NP2003 and HP2003. The quality of URD and UFD discretizations vary more widely being very good in some cases but doing poorly in others. Although UFD obtains the highest average for all datasets (0.4940), TUBE is a very close second (0.4927) providing better results on the more difficult informational datasets (TD2003 and TD2004). TUBE seems to be a very good choice for our active selection method, since we are assuming that we do not have any training set and that we want to build one from scratch. In such a setup, it is not easy to evaluate several discretization methods beforehand. TUBE seems to provide good results for datasets with very diverse characteristics such as the HPs and TDs.

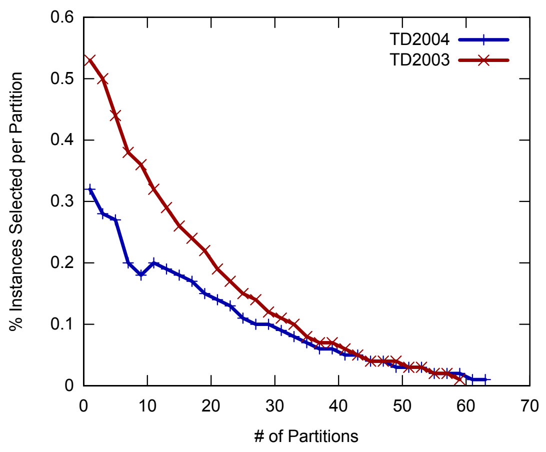

Figure 4.2. % of instances selected per partition as we vary the number of partitions

3.3.2 Partitioning

Another key element of ARLR is the partitioning of features used to delay the algorithm's convergence and possibly increase sample diversity. As explained in section 3.2, we partition the dataset vertically, creating subsets or partitions that contain all unlabeled instances but only a group of the features. All features are ranked based on their χ^2 measure in relation to one another and then spread on the partitions uniformly. ARLR is then used to select instances from each of these partitions in turn and then all selected samples are gathered in a new training set that contains all features. As we have seen, this process allows ARLR to select more instances than if no partitioning is used. It possibly also increases the diversity of the selected samples since they are chosen based on distinct feature sets. In other words, the instances are selected for their distinctiveness within each feature group, which may increase diversity overall.

The partitioning process involves a trade-off. As we increase the number of partitions, less features are put into each partition which potentially allows for more diversity in the final selected set; but less features per partition also means less *information* about each instance based on which ARLR must select samples. As the number of features per partition reduces, the overall quality of the instances selected by ARLR may be affected, which in turn may reduce the quality of the final selected set. To better understand the effects of partitioning, we show some results from experiments using TD2003 and TD2004 (the other datasets have similar results and are omitted for simplicity).

We can see this effect in figures 4.2 and 4.3. Figure 4.2 shows the amount of instances selected per partition (as a percentage of the full unlabeled set) as the number of partitions increase and, thus, the number of features per partition decreases. As we

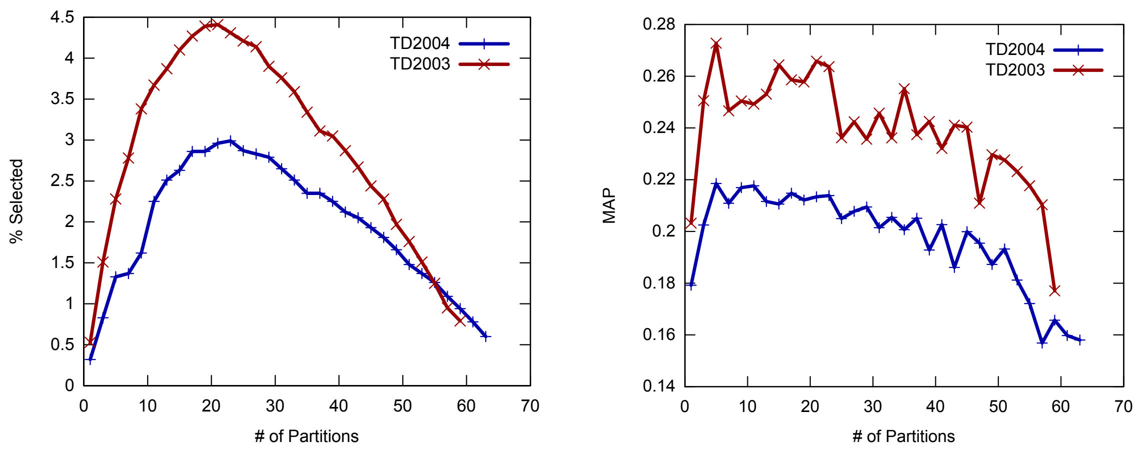

Figure 4.3. % Selected samples vs. # of Partitions (left) and MAP vs. # of Partitions (right)

use more partitions, less instances are selected *per* partition, since there are less features for ARLR to generate rules from. To the left of Figure 4.3, we see the proportion of samples selected from all partitions as more partitions are used. As we use more partitions, we increase the final size of the selected set until it reaches a point where the total number of selected samples starts decreasing (at around 21 partitions for TD2003 and 23 for TD2004: at this point, each partition has around 3 features). Eventually, only one feature is used per partition (at 59 partitions for TD2003, which has 5 empty features, and 64 partitions for TD2004), and the total number of selected samples drops to almost the original number obtained by using ARLR with no partitioning (i.e. using only one partition; see table 4.1).

On the graph to the right of Figure 4.3, we see a plot of the MAP of the ranking produced by RLR using the training sets produced by ARLR as we vary the number of partitions. The MAP for both datasets peak at 5 partitions (12 features per partition) and then varies considerably, going slowly down and finally dipping to its minimum at the maximum number of partitions. At the end of the run, where each partition contains only one feature, ARLR selects more samples than at the beginning (i.e. 1 partition containing all features), but the MAP obtained is lower there, indicating that the few samples selected using all features are more informative than the set selected based only on one feature. The effect of the information loss that occurs when a very small number of features is used to select samples can also be seen in the jagged line of the MAP for both datasets from 21 partitions on. From 21 to around 33 partitions, each partition has 2 or 3 features. From 33 to the end, each partition has either 2 features or 1 feature. The variation of the MAP indicates that some *combinations* of features are more valuable than others: as we split the features in different ways - reducing the number of partitions with 2 features and increasing the number of partitions with only 1 feature -, sometimes we get one or more groups of features that allow ARLR to select more informative samples, increasing the final MAP. On the next round, using more partitions, these combinations change and the resulting MAP again drops. Eventually, almost all partitions contain only one feature and there are almost no useful combinations, making the MAP drop steadily and sharply.

From these results, we can see that we need to provide ARLR with enough features to be able to uniformly select informative samples from each partition. If we have too many features per partition, then the selected set may be too small, but if we have too few features per partition, then ARLR may select informative samples from some partitions but not from others. Determining the number of partitions *a priori* is difficult, since different datasets will contain different numbers of features and, more importantly, more or less very informative features. But from these experiments we can see that the resulting MAP is relatively stable when using from 3 to 13 features per partition (i.e. 5 to 20 partitions for the LETOR 3.0 datasets). This relative stability is a direct consequence of the ranking of features, since by using it we try to make sure that very informative features are balanced across the partitions; that is, each partition contains one or more "good" features which allows ARLR to select quality samples from all partitions.

3.3.3 Using the selected samples with other L2R methods

Once the instances are actively selected by ARLR, it is possible to use other supervised learning algorithms to rank the test sets. Of course, different algorithms will find the instances selected by ARLR more or less "informative". Active learning methods currently proposed in the literature are inherently algorithm-specific as illustrated by the SVM-specific techniques discussed in Section 4 of Chapter 3. For instance, the approach proposed in Yu [2005] selects, at each round, the samples that minimize the support vector margin. What if we run an SVM ranking algorithm using the selected instances as training[2]?

Table 4.4 shows the MAP resulting from running SVMRank[3], proposed by Joachims [2002] and discussed in Section 6 of Chapter 2, using the examples selected by ARLR as training sets. The column "ARLR" repeats the results from Table 4.2 for reference. "SVMS" shows the MAP obtained from running SVMRank using the

[2]One reason to use RankSVM is that it is one of the best performing methods on LETOR 3.0.
[3]Best parameters were determined using cross-validation on the training sets.

Table 4.4. MAP for SVM using selected samples, BM25 and Random Baselines

	ARLR	**SVMS**	**SBM25**	**Random**	**G%**
TD2003	0.2728	0.2881	0.1568	0.1417±0.0285	83.74
TD2004	0.2185	0.2009	0.1335	0.1687±0.0145	24.11
NP2003	0.6960	0.6524	0.6587	0.5739±0.0237	-0.95
NP2004	0.5499	0.6098	0.5811	0.5787±0.0329	4.94
HP2003	0.7411	0.7226	0.7090	0.5798±0.0592	1.92
HP2004	0.6168	0.6661	0.6731	0.5406±0.0357	-1.04

samples selected by ARLR (see Table 4.2 for other information such as the number of instances selected, etc.). Again, we ran two baselines for comparison: "SBM25" contains the results from running SVMRank with the same amount of samples as selected by ARLR, but picked by their BM25 value. "Random" shows the average of 20 runs where the instances are randomly selected and includes the confidence interval for a 95% confidence level. "G%" indicates the gain of SVMS over the best baseline (i.e. either SBM25 or Random).

From the results we can observe that SVMS fares very well on the TD2003, TD2004 and NP2004 datasets as compared to the baselines. For the other datasets, it almost ties with the SBM25 baseline. This may be due to the fact that the BM25 method selects a high proportion of relevant instances which are extremely rare in the original NP and HP datasets (column "R%" of Tables 4.1 and 4.2). This is one of the difficulties in ranking these datasets and both ARLR and the BM25 selection methods may help as they tend to select a larger proportion of relevant samples. Though these results can be improved, they illustrate that our method chooses informative instances that are useful even for completely different algorithms.

3.3.4 Comparing the results with LETOR baselines

To put ARLR results in perspective, Table 4.5 shows the MAP results for ARLR and those for four supervised algorithms from the LETOR 3.0 published baselines[4] (that use the full training sets). The producers of the LETOR datasets have published results containing precision, MAP and NDCG obtained using 12 L2R algorithms on the LETOR 3.0 datasets (see Qin et al. [2010b]). We chose to show the results for four algorithms: RankBoost ("RBoost" in Table 4.5), FRank, Regression ("REG") and SVMRank ("SVMR"). The first two were selected mainly because, similarly to our method, they use non-linear ranking functions, so they all lie in the same class of algorithms, making this a more fair comparison. RankBoost achieves very good results,

[4] http://research.microsoft.com/en-us/um/beijing/projects/letor/letor3baseline.aspx

Table 4.5. MAP for ARLR and LETOR Baselines

	ARLR	**RBoost**	**FRank**	**REG**	**SVMR**
TD2003	***0.2728***	*0.2274*	*0.2031*	*0.2409*	*0.2628*
TD2004	*0.2185*	0.2614	0.2388	*0.2078*	0.2237
NP2003	**0.6960**	0.7074	*0.6640*	*0.5644*	*0.6957*
NP2004	*0.5499*	0.5640	0.6008	*0.5142*	0.6588
HP2003	***0.7411***	*0.7330*	*0.7095*	*0.4968*	*0.7408*
HP2004	*0.6168*	0.6251	0.6817	*0.5256*	0.6675
Avg.	*0.5158*	0.5197	0.5163	*0.4250*	0.5415

beating all other 11 algorithms in 2 of the 6 datasets (TD2004, NP2003). FRank has average results if compared to the other algorithms, Regression obtains a lower average then the other algorithms, although it still beats some of the them in some datasets. We use the results of these three algorithms as high, medium and low watermarks to show that ARLR obtains competitive results selecting only a very small fraction of each dataset. We also include the published SVMRank results, since in the next chapter we propose an extension of our active learning framework using this algorithm.

In Table 4.5, ARLR results that appear in *italic* are equal or better than one of the four baselines. Numbers in **bold** indicate that the result is equal or better that 2 or 3 of the baselines. Finally, the results in ***bold and italic*** are equal or better then all four baselines. We indicate which results from the baselines were matched or surpassed by showing them in *italic*. From Table 4.5 we can see that ARLR beats the chosen supervised baselines in TD2003 and HP2003. It also obtains very good results for NP2003, beating 2 baselines and almost reaching the performance of the 3^{rd} (RankBoost). Overall, ARLR's average performance across the 6 datasets is only 1% below RankBoost but using on average 98% less training. SVMRank provides very robust results on all datasets, obtaining a higher average; still ARLR results are better in 3 of the datasets. For further performance comparisons, Table 4.6 provides the NDCG@1, @5 and @10 using the same baseline algorithms and markup scheme.

3.3.5 Comparison with other active learning methods

We implemented the method proposed by Donmez and Carbonell [2008] (and described in Section 4.2 of Chapter 3) in order to be able to compare our algorithm to another active learning to rank strategy. We call it "Donmez" for short. The Donmez method is based on the assumption that those instances that change the current model the most are more "interesting". Instead of retraining the learner on each and every unlabeled sample one by one, Donmez proposes calculations to estimate the change in the current learner for two algorithms: SVMRank and RankBoost. We implemented the SVMRank

Table 4.6. NDCG for ARLR and LETOR Baselines: ARLR vs. Rankboost and FRank (a), ARLR vs. Regression and SVMRank (b)

(a)

	ARLR			**RankBoost**			**FRank**		
	@1	@5	@10	@1	@5	@10	@1	@5	@10
TD2003	***0.4600***	**0.3280**	**0.3336**	*0.2800*	*0.3149*	*0.3122*	*0.3000*	*0.2468*	*0.2690*
TD2004	**0.4533**	**0.3547**	**0.3332**	0.5067	0.3878	0.3504	0.4933	0.3629	*0.3331*
NP2003	**0.5867**	***0.7850***	**0.7918**	0.6000	*0.7818*	0.8068	*0.5400*	*0.7595*	*0.7763*
NP2004	*0.4133*	**0.6546**	*0.6805*	0.4267	*0.6512*	0.6914	0.4800	0.6870	0.7296
HP2003	***0.7200***	**0.7809**	**0.7982**	*0.6667*	0.8034	0.8171	*0.6533*	*0.7780*	*0.7970*
HP2004	**0.5200**	*0.6981*	*0.7111*	*0.5067*	0.7211	0.7428	0.6000	0.7486	0.7615

(b)

	ARLR			**Regression**			**SVMRank**		
	@1	@5	@10	@1	@5	@10	@1	@5	@10
TD2003	***0.4600***	**0.3280**	**0.3336**	*0.3200*	*0.2984*	*0.3263*	*0.3200*	0.3621	0.3461
TD2004	**0.4533**	**0.3547**	**0.3332**	*0.3600*	*0.3257*	*0.3031*	*0.4133*	*0.3240*	*0.3078*
NP2003	**0.5867**	***0.7850***	**0.7918**	*0.4467*	*0.6423*	*0.6659*	*0.5800*	*0.7823*	0.8003
NP2004	*0.4133*	**0.6546**	*0.6805*	*0.3733*	*0.6135*	*0.6536*	0.5067	0.7957	0.8062
HP2003	***0.7200***	**0.7809**	**0.7982**	*0.4200*	*0.5463*	*0.5943*	*0.6933*	0.7954	0.8077
HP2004	**0.5200**	*0.6981*	*0.7111*	*0.3867*	*0.6130*	*0.6468*	0.5733	0.7512	0.7687

variation only. The method starts with an initial labeled set comprised of 15 randomly picked documents *per query*, plus exactly one relevant document (this amounts to 1.6% of the unlabeled sets for all LETOR 3.0 datasets). From then on, the algorithm chooses 5 documents per query per round. Thus, this is a round-based method where 0.5% of the documents in the unlabeled set are selected at each round. In Donmez and Carbonell [2008], Donmez presents results for two of the LETOR 2.0 datasets, running his method for 25 rounds. Since Donmez uses randomly selected instances as its initial sets, results reported are averages for 10 runs for each fold (50 runs total).

Figure 4.4 shows the results of Donmez's method compared to ARLR and SVMS (SVMRank using the instances selected by ARLR as training). Notice that both ARLR and SVMS use only the very small set selected by ARLR and are shown in the plots as lines for clarity (both should be only a point in the extreme left of each plot). The percentage of instances selected at each round for Donmez is shown in the x-axis, while the percentage used by ARLR and SVMS appear in the key.

Observe that ARLR beats Donmez *in all rounds* on the three 2003 datasets. In contrast, Donmez is better in all rounds on NP2004 and HP2004. For TD2004, Donmez reaches ARLR's performance on the 3^{rd} round (when it has selected 3.12% of the instances in the unlabeled set). SVMS does very well on TD2003 and is slightly better than Donmez up to the 3^{rd} round on HP2003, but fares worse on the remaining datasets. Overall, ARLR is a strong method, selecting very small training sets

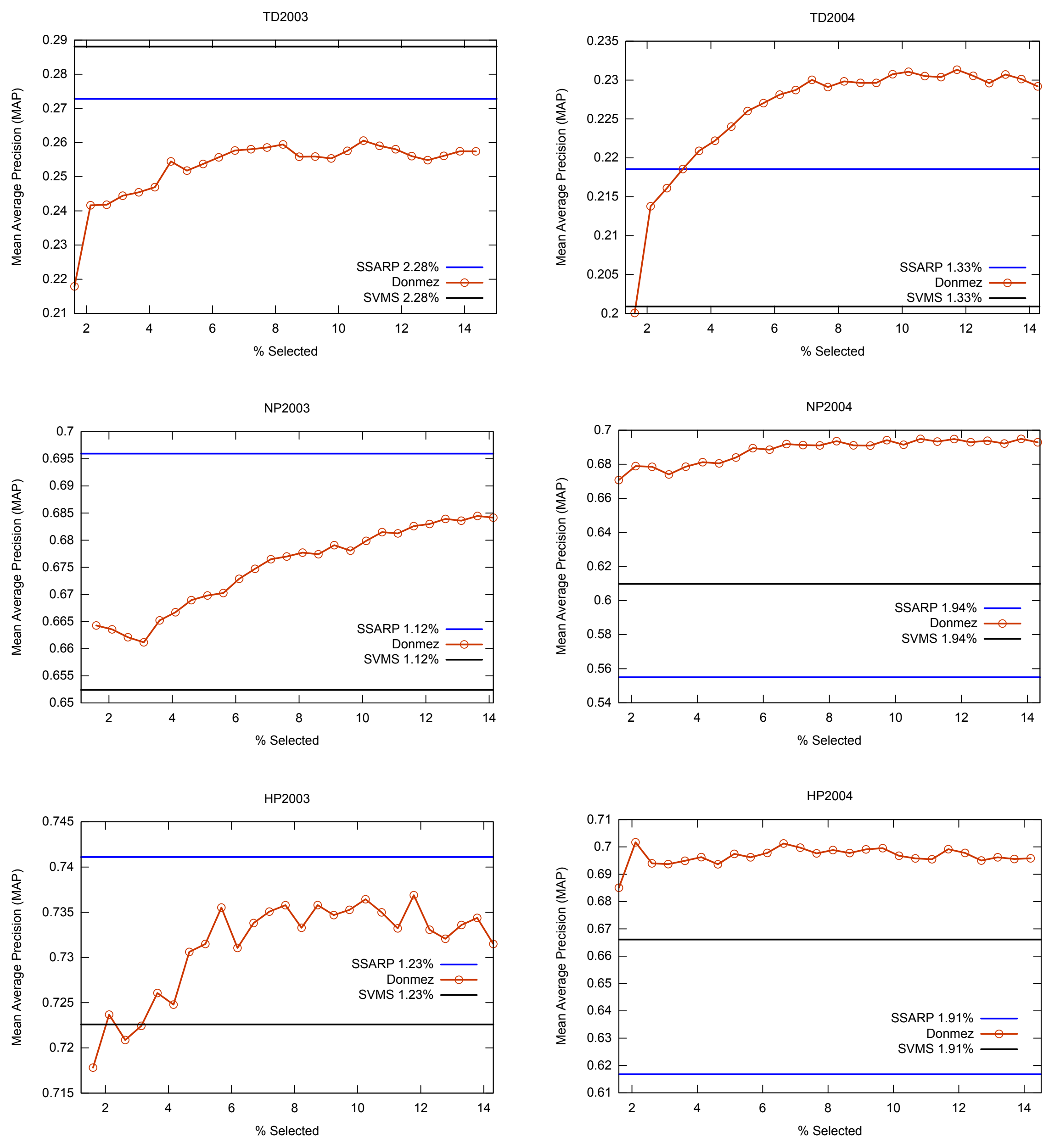

Figure 4.4. Comparison of ARLR, SVMS and Donmez: MAP vs. % of samples selected from unlabeled set for 25 rounds

that provide competitive results when compared to supervised baselines and an active learning method proposed in the literature.

4 Summary

We have proposed a rule-based active learning method that is practical and effective, selecting very few documents to be labeled and yet offering competitive results when compared to established supervised algorithms using the complete training sets. Since the method does not depend on an initial labeled set, it can be easily used in real-life applications where labeling many documents can be prohibitively expensive or unpractical.

As we have seen, the proposed method obtains results that are better or on the same level as those obtained by established supervised methods (using the full training sets) in some datasets (i.e. TD2003, TD2004, HP2003, NP2003) but selecting and using only a *fraction* of the original training sets. This is one of the stated goals of an active learning method: to be able to select a small and yet effective training set, reducing the cost associated to labeling documents and producing training sets.

Although the proposed method is very effective for some datasets, it does not fare so well in others (i.e. HP2004, NP2004). By providing a natural stop condition, the method is very simple to use, since it can be run on a dataset until it stops and only *then* we can use cross-validation to evaluate the quality of the selected set. But if the quality achieved is not the desired one and/or if there's still labeling budget to spend, the method does not provide a simple way to select and label more samples. In the next chapter we propose an extension to the method that overcomes this limitation, allowing us to expand and improve the selected training set.

Chapter 5

Expanding Selection with Query-By-Committee

1 Introduction

In this chapter we propose a new hybrid active learning technique that uses association rules and a query-by-committee (QBC) procedure to obtain from an unlabeled set a small and yet highly effective training set. This hybrid method combines the best of two different active learning strategies while trying to overcome their drawbacks. On one hand, although ARLR produces very reasonable results (i.e., very small and effective training sets), it does not provide a simple way to select more instances (and possibly improve the ranking quality), even if there is labeling budget available, since it stops selecting new instances for labeling when it judges that no other candidate has useful information to be incorporated into the training set.

By incorporating a QBC procedure we are able to select more instances using a completely different selection mechanism. On the other hand, traditional QBC strategies use a semi-random initial set, which has to be large enough to work well (otherwise the performance may be severely hurt, especially in the initial rounds, as shown in the results section). By using the sets selected by the rule-based active learning to feed the QBC round-based active learning our method maximizes performance from the start, allowing for any size of labeling budget to obtain very good results. This is an important characteristic for an active learning method, since in a real-world scenario there is no simple way to evaluate the quality of the selected sets: only after the sets are actually selected and labeled (i.e. after the labeling budget is spent) we can use cross-validation to evaluate the sets' quality. If the labeling budget is small, our method obtains very good results by converging fast to high-quality rankings (as a function of the size of the selected set). If the budget is bigger or more flexible, our method still selects a high quality training set. We believe this work is an important contribution as the method is both practical and effective.

The proposed hybrid method works in two stages: in the first one, ARLR is used to select a very small initial labeled set. In the second stage, this initial set is used to train three learning to rank functions. Then each query in the unlabeled set is ranked using these functions and those documents that have the highest variation in the distinct rankings produced by these functions are deemed the most discordant and selected. This second stage - the QBC stage - is repeated iteratively until a target training size is achieved or the labeling budget met. The concept behind this iterative process is that the documents for which the rankings vary the most are those that the rank learning algorithms most disagree about. The degree of disagreement can serve as an estimate of the informational value of each document to the learning algorithms, as stated by Seung et al. [1992]. In the QBC stage of our active learning method, we use three algorithms as our committee: SVMRank, RankBoost and Rule-based Learning to Rank (RLR).

We compare our method against five baselines: random sampling, the active method proposed in Donmez and Carbonell [2008], which we call Donmez, a modified version of Donmez using the first stage training data, the supervised SVMRank results obtained using the complete training sets and the ARLR active learning results presented in Silva et al. [2011] and in Chapter 4. Experiments were run using the LETOR 3.0 web datasets and selecting up to 15% of each original training set for comparative purposes. As we will see in the experiments subsection, our method obtained very good results selecting as little as 5% of the original unlabeled sets. The results surpass, in most cases, even state-of-the-art supervised algorithms using the complete training sets.

2 Stage 1: Active Learning using Association Rules

Stage one of our hybrid method is ARLR as described in the last chapter. That is, we use ARLR with 5 partitions and 10-bin TUBE discretization, obtaining very small initial sets (see Table 4.2 for the sizes of the initial labeled sets obtained by ARLR for each of the datasets). Instead of randomly selecting the initial labeled sets, as is usually done in QBC-based methods, we choose to use ARLR because, as we have seen, it selects small but effective sets.

3 Stage 2: Query-By-Committee

3.1 QBC Active Learning

Stage 1 of our method selects a very small initial set that can be used as a training set for our QBC iteractive active selection stage. The concept of using a committee of learners to identify "interesting" data instances is well known (see Baum [1991]; Schapire [1989]; Seung et al. [1992]). The basic idea is to use an ensemble of learners or models to classify (or rank) an unlabeled set and those data instances that the diverse models most disagree about are deemed the most "informative" and selected for labeling. Seung et al. [1992] calls this process "incremental learning" where a training algorithm is used to produce a committee of $2k$ learners and a query algorithm selects a sample that is classified positive by half of the committee and negative by the other half. They show that, for the Gibbs training algorithm, in the $k \to \infty$ limit each query bisects the version space, maximizing the information gain.

More general methods have been proposed and successfully used such as query-by-bagging and query-by-boosting (see Abe and Mamitsuka [1998] and Schapire [1989], respectively). In the bagging approach, different models are produced using the same learning algorithm trained on partitions created by uniformly sampling the current labeled set. Boosting uses a more sophisticated, round-based re-sampling method where the sampling distributions (or instances' weights) are varied at each round so as to focus the learned models on the parts of the training data that previous learners did poorly on. The bagging concept is immediately applicable to active learning, where the instances selected at each round are those which the diverse models most disagree about. In an L2R scenario, this measure of disagreement could be, for instance, the Kendall's τ rank correlation coefficient or, as proposed in Cai et al. [2011], a vote entropy (see Dagan and Engelson [1995]) variation adapted for pairwise ranking. These metrics would allow the measurement of the disagreement of the learners in ranking *each query*. This means we would need to select whole queries (i.e. perform *query-level* selection), instead of sampling documents from the unlabeled set. This may be fine for datasets containing few documents per query but doing query-level selection on the LETOR 3.0 datasets would imply selecting at least 1,000 documents per round.

Our method uses a simpler ensemble approach to QBC where different *algorithms* are used to produce diverse rankings at each round. Thus, we train three distinct algorithms using the same training data (the labeled data available at each round) and rank the remaining of the unlabeled set using these three learners. Then, for *each document* of each query, we calculate a simple metric (explained below) to determine which documents of that query the three learners most disagree in ranking. At each round, we select the 5 documents from each query which have the highest value for the disagreement metric described below. To rank the unlabeled sets, we use three algorithms in our committee: SVMRank (Joachims [2002]), RankBoost (Freund et al. [2003]) and Rule-based Learning to Rank (RLR) (Veloso et al. [2008]). These algorithms are trained using the labeled set gathered so far and then used to rank the remaining instances in the unlabeled set.

We sample 5 documents per query per round simply to be able to compare our results with our main baseline which was proposed by Donmez and Carbonell [2008]. As we will see in the results subsection, this works fine for the LETOR 3.0 datasets, which have few queries and many documents per query. But our method could be easily adapted to either: a) perform query-level selection for datasets with many queries and few documents per query; or b) perform first a query-level selection and *then* a document level selection in datasets with many queries and many documents per query. Option b) is probably the most reasonable as selecting all documents of a query, even in datasets with few documents per query would likely introduce noisy instances into the selected set, possibly hurting the results. Furthermore, option b) provides greater flexibility, since we could tune the number of queries selected per round as well as the number of documents selected per query/round, adapting the number of selected samples/round to a number compatible to the characteristics of the dataset at hand and the labeling budget.

3.2 Measuring Rank Disagreement

At each round, the unlabeled documents of each query are ranked by the members of the committee using models trained on the labeled sets accrued so far. If we were doing query level selection, as in Cai et al. [2011], we would need to measure the disagreement of each ranking on the *query level*. Instead, our method works at the document level: that is, we want to measure the disagreement among the learners about the *ranking of each document*. We aim to select those documents that vary the most in their positions in each ranking. One simple metric would be to calculate the variance or standard deviation of each document's positions. For example, if we have three learners and document d_i is ranked in positions 10, 15 and 20 of each ranking, this document would have a ranking variance of 25 (or σ of 5). The problem of this approach is that it gives the same importance to documents no matter whether they appear at the top or at the bottom of the rankings. For instance, document d_j appearing in positions 110, 115 and 120 would have the same variance of 25. In L2R we are usually much more concerned with the very top of the ranking Granka et al. [2004] and, thus it would seem preposterous to assign the same value of disagreement to documents d_i and d_j in the example above. A simple solution is to use the Coefficient of Variation, which is a normalized measure of dispersion. The CV is defined as σ/μ. Now, documents far down the rank will need much more varying rankings in order to be selected. For example, a document d_k with rankings 110, 200 and 220 will have almost the same CV as document d_i with rankings 10, 15 and 25.

4 Experimental Evaluation

4.1 Experimental Setup

In our active learning scenario, we consider the original training sets of each dataset as unlabeled sets from which we select instances to be labeled. The label of these instances is not used until they are selected and we assume that a human annotator evaluates and labels each selected document. Our method, ARLR-QBC, is run in each fold of each dataset in the following manner:

First Stage (ARLR): In the first stage, the unlabeled and test sets are discretized using the Tree-based Unsupervised Bin Estimator (TUBE) proposed in Schmidberger and Frank [2005]. TUBE is a greedy non-parametric density estimation algorithm that uses the log-likelihood measure and a top-down tree building strategy to define varying-length bins for each attribute in the dataset. All sets are discretized using 10 varying-length bins. Then the unlabeled set is partitioned into 5 vertical partitions as proposed in Silva et al. [2011] and in Chapter 4. Each partition contains all unlabeled instances, but only a group of each instance's features. ARLR is run on each partition, selecting instances based on distinct feature sets. This process is done in order to increase the number of selected instances, since for each partition the algorithm will choose those that are most informative given the features in that partition. It also improves the diversity of the samples, as they are selected based on distinct characteristics. The final product of this first stage of the method is a very small labeled set that we use as an initial training set for stage 2 (the x in the ARLR $x\%$ baseline in the results shows the size of the initial sets selected).

Second Stage (QBC): The first step in stage 2 is to determine the best parameters for the algorithms used in the committee. To do that, we split the tiny labeled sets produced in stage 1 into 5 parts and use 3 of those as training e 2 as test sets. We then run the algorithms varying the parameters and choose the parameter(s) that gives us the best MAP. Stage 2 progresses in rounds where, at each round, 5 instances are selected *per query*. This number is used so that we can compare our method with that presented in Donmez and Carbonell [2008], one of our baselines. At the first round, we use the labeled set produced in stage 1 to train our 3 supervised algorithms: SVMRank, RLR and RankBoost. The models produced are used to rank the unlabeled sets (which, at this point, are the original training sets *minus* the instances selected in stage 1). Then, for each query, the rankings are evaluated and the metric described in subsection 4.1 (CV) is calculated for each document. The 5 documents with the highest CV value are then selected to be labeled. At the end of the round, 5 times the number of queries in the unlabeled set need to be labeled by the human annotators (in TD2004, for instance, $5 \times 45 = 225$ documents are selected per round; for all datasets the number of documents selected per round represent 0.5% of the unlabeled sets). Once these documents are labeled, they are inserted into our labeled set and removed from the unlabeled set. As another round starts, the augmented training sets are used to train the three committee members and the process described above is repeated. This procedure can be repeated as many times as desired or until the labeling budget is exhausted.

4.2 Baselines

RR-Donmez: Our main baseline, which we call Donmez, is an implementation we did of the method proposed by Donmez and Carbonell [2008] and described in Chapter 3, Section 4.2. The Donmez method is based on the assumption that those instances that change the current model the most are more "interesting". Instead of retraining the learner on each and every unlabeled sample one by one, Donmez proposes calculations to estimate the change in the current learner for two algorithms: SVMRank and RankBoost. We implemented the SVMRank variation only. The method starts with an initial labeled set comprised of 15 randomly picked documents *per query*, plus exactly one relevant document. From then on, the algorithm chooses 5 documents per query per round. To make the comparison easier, our method also selects 5 documents per query per round, and we also run our experiments for 25 rounds. Since RR-Donmez uses randomly selected instances as its initial sets, results reported are averages for 10 runs for each fold (50 runs total).

ARLR-Donmez: In this variation of Donmez's method the initial set used is the one selected by ARLR. This baseline is intended to put ARLR-selected sets in perspective and pitch QBC vs. Donmez.

R-Random: In the random baseline, all instances are randomly selected. The amount of instances selected is exactly the same as Donmez's RR method: it starts

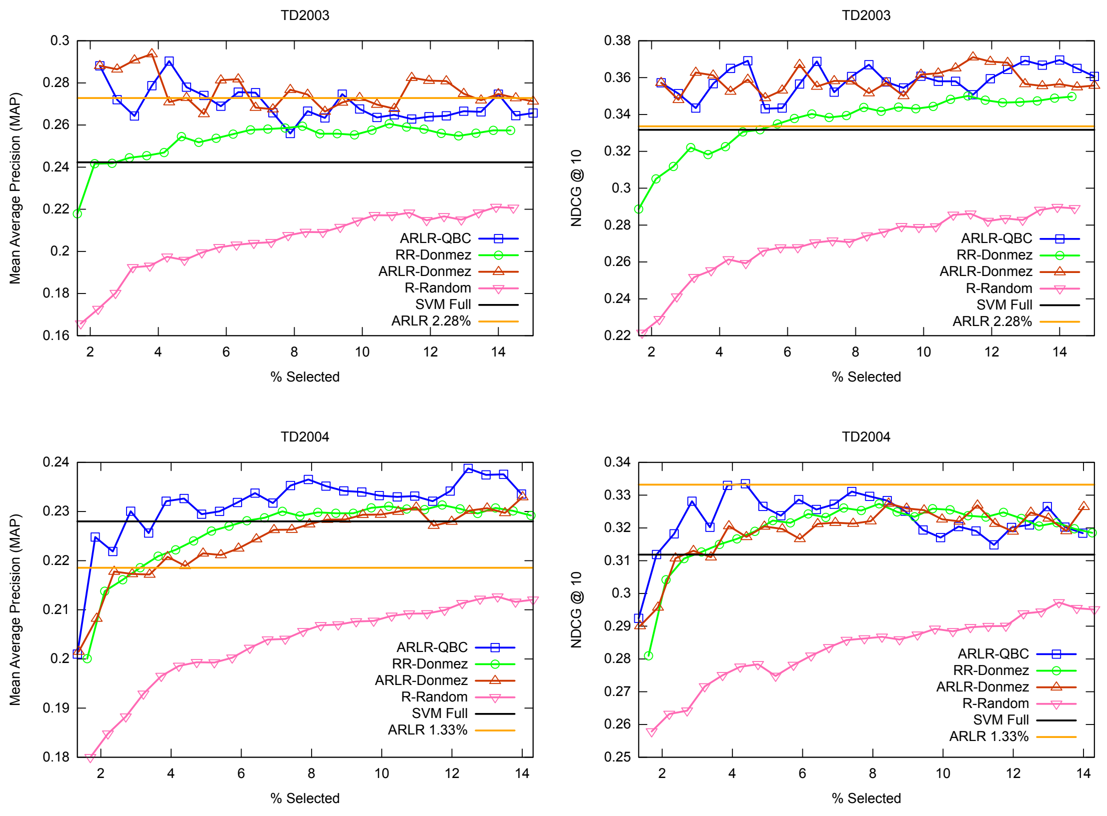

Figure 5.1. TD2003 and TD2004: ARLR-QBC and baselines comparison. MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

with an initial random set of 16 instances per query (1.6% of each set) and selects 5 random instances per query per round (or 0.5% of each set). Results reported are averages for 10 runs for each fold (50 runs total).

SVM Full: This is the value obtained by running SVMRank using the complete training set. We include this baseline as a line in each graph (although it actually uses 100% of the training set) as a reference to put our results and the other baselines in perspective.

ARLR x**%**: This line indicates the result obtained by using ARLR to select instances and RLR to rank the test set (these are the results that appear in Chapter 4, Table 4.2). The x value indicates the proportion of the original training sets that was selected by ARLR. Again, we present this result as a line for readability (as this result should actually be a point in the extreme left of each plot).

4.3 Results

Each round-based plot line is named according to the method used in each stage of the process. The first stage is the selection of the initial training sets. ARLR is the rule-based active sampling algorithm described in Section 3.2. RR (Random with 1 Relevant) is the method used by Donmez that selects 16 random samples with exactly one relevant. R (Random) selects 16 random samples. In the second stage (round-based selection) QBC indicates the method described in subsection 4, Donmez designates the method proposed in Donmez and Carbonell [2008] and Random is a random sampling. Thus "ARLR-QBC" is the method proposed in this chapter, in which ARLR is used in the first stage to select the initial training sets and QBC is used to select samples per query. For all results and baselines, except for "ARLR $x\%$", after the instances are selected at each round, SVMRank is used to rank the test set in each fold (using the currently selected sets as training), producing the final MAP and NDCG@10 metrics. "ARLR $x\%$" uses instead the RLR supervised ranking method described in Chapter 2, Section 5 and Veloso et al. [2008].

4.3.1 Informational Datasets

Figure 5.1 shows the performance of our method against the chosen baselines on TD2003 and TD2004. These are the most relevant datasets, since they are built using the more "general" Topic Distillation queries (in contrast to the more "specific" navigational queries used to build the HP and NP datasets). The plots show on the y-axis the MAP (left) and NDCG@10 (right) obtained at each round for each method. The x-axis indicates the percentage of the unlabeled set selected at each of the 25 rounds, plus the results obtained using only the initial training set (therefore, each plotted line has 26 points). The ARLR-based lines in the plot start at 2.23% and 1.33% for TD2003 and TD2004, respectively. The size of the initial training sets selected using ARLR varies for each dataset, since the method naturally converges, as explained before. The RR and R-based lines always start at 1.6% for all datasets considered.

TD2003 is exceptional in our results, as the ARLR-selected initial set performs very well in this dataset using both SVMRank and ARLR 2.28%. Using only the initial dataset (comprised of 2.28% of the unlabeled set) SVMRank obtains a MAP of 0.2881 and ARLR (with RLR) a MAP of 0.2728, passing not only SVMRank using the full training set (SVM Full), but also (in the case of SVMRank) all supervised baselines published by the LETOR producers. With such an exceptional initial result, there is little room for improvement. In fact, although both ARLR-QBC and ARLR-Donmez do achieve slightly better results in a few rounds (ARLR-QBC on round 4 with MAP 0.2907 and ARLR-Donmez on round 3 with MAP 0.2936), in the remaining rounds the results obtained go up and down with a slight downward trend (the NDCG@10 results remain mostly the same, but also jump up and down). Remember that Donmez selects

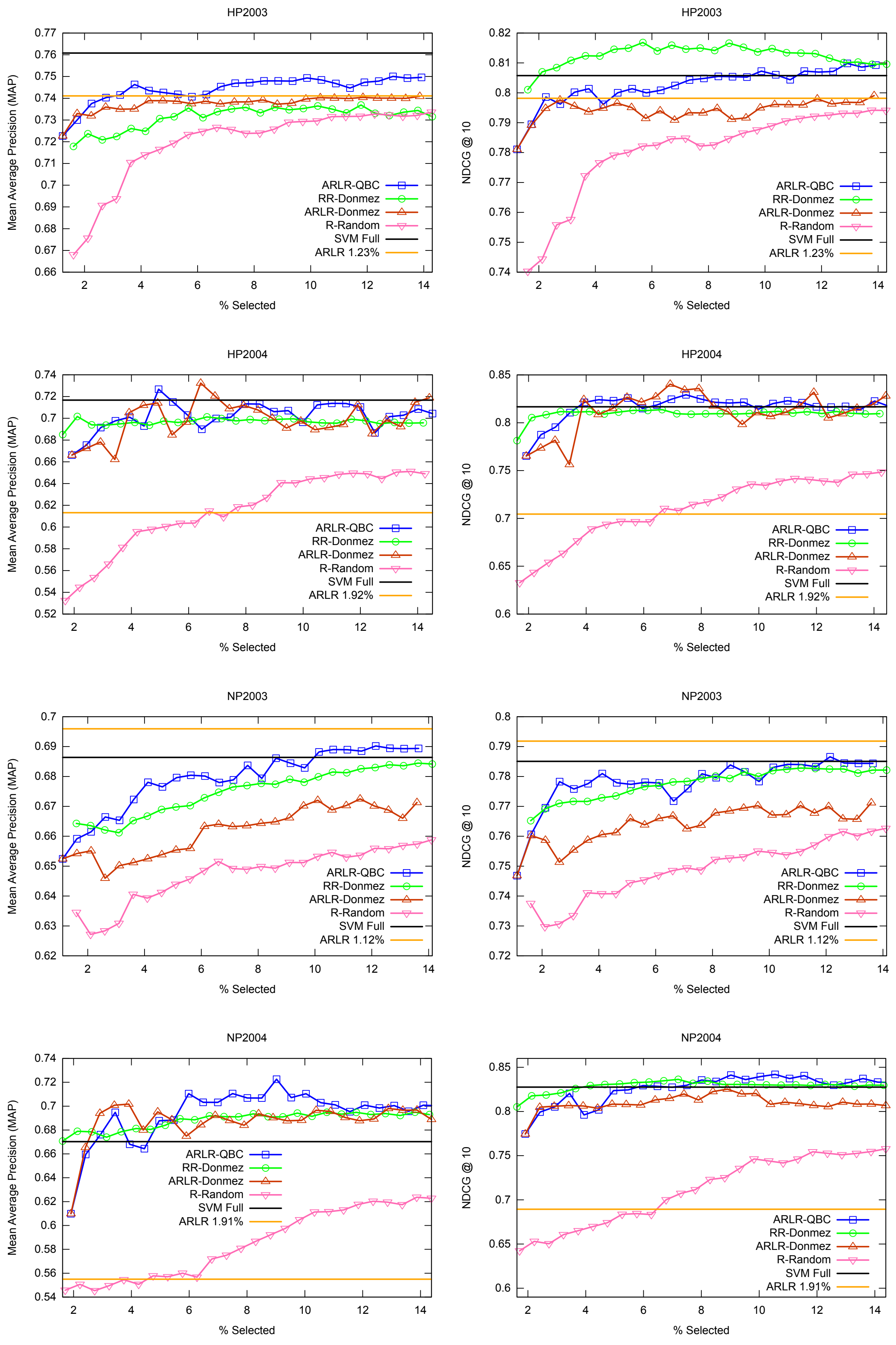

Figure 5.2. HP2003, HP2004, NP2003 and NP2004: ARLR-QBC and baselines comparison. MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

the instances that are estimated to change the learned model the most; in this situation where the set selected by ARLR is already very good, this strategy backfires in some rounds, changing the model for worse. The same happens with QBC, although this strategy does not explicitly target model modification. It seems that once a very good training set is obtained it is hard to select more instances without adding some noise to it. Although RR-Donmez obtains good results in this dataset, beating SVMRank Full on both metrics, ARLR-QBC results are still significantly better than RR-Donmez with 95% confidence level up to the 9^{th} round for the MAP (6.86% selected) and up to the 6^{th} round for the NDCG@10 (5.34% selected).

TD2004 shows a different picture, with our method also surpassing Donmez-based baselines in the initial rounds for both metrics, but starting much closer. With this dataset, our method is significantly better than Donmez's with 95% confidence level up until the 6^{th} round (4.37% selected) for both metrics. For this dataset, the Donmez selection method starting with ARLR-selected sets (ARLR-Donmez) performs roughly equal to the original method (RR-Donmez). We can see that ARLR does not select such a good initial set as in the case of TD2003, but QBC selection performs very well bringing the metrics on par with SVMRank using the complete set (i.e. SVM Full) when the selection reaches only 4% of the unlabeled set. Also notice that the random baseline is surpassed by not only our method, but also by all the other baselines in all datasets.

4.3.2 Navigational Datasets

In Figure 5.2, we can see that our method performs very well in HP2003, NP2004 (MAP) and NP2003 (both metrics) and roughly similar to Donmez on HP2004 or in all datasets but HP2003, if we consider the NDCG@10. The exception is the NDCG@10 result for HP2003, where Donmez performs exceptionally well. Also notice that RR-Donmez has a better start than our method in all but HP2003. These datasets have a particularly low proportion of relevant documents (see Table 2.1). This is natural, given that these datasets are Named Page (NP) and Home Page (HP) distillations, i.e., there is usually only one relevant document per query. The RR selection method used by Donmez *always* inserts 6% of relevant instances into the initial sets (1 per query), while ARLR selects a varying proportion of relevant documents. This could be an explanation to Donmez's better performance in the initial rounds on the HP and NP sets, as relevant documents are essential for SVMRank to produce pairwise comparisons.

It is important to observe that RR is not a practical, but a simulated initial set selection method. Donmez's papers Donmez and Carbonell [2009, 2008] assume that these initial sets are available without explaining how they are obtained. This may be the case in certain situations, but if we consider an active learning scenario where it is necessary to build a training set from scratch (which is what ARLR was designed to do), then RR is unrealistic and we need to devise a practical method to select a RR-like initial set. We propose practical variations of RR in the next section and analyze the results obtained from new experiments.

4.4 Analysis

4.4.1 Making RR "Real"

RR selects 16 instances per query randomly with exactly one being relevant. If we do not have *any* labeled instances, we must devise a way to pick instances from the unlabeled set and label them so that we choose approximately one relevant instance for each 15 non-relevant labeled. For most datasets, randomly picking instances and labeling them would be too costly (i.e. we would need to label tens or hundreds of instances before finding a relevant). We can get much closer to our target if we order the instances for each query using one or more features that we know have values highly correlated to relevance. BM25 could be used, for instance, but it is not necessarily the best feature for all datasets. Let's assume that, even without having labeled instances, we have a good idea of which feature to use to maximize the chances of finding at least one relevant instance for every 16 we label. To approximate RR to reality, we use a synthetic method and simulate an "oracle" that gives us the best feature to use to rank the unlabeled set in such a way that we can find a relevant instance as early as possible. This oracle uses the labeled sets (which are not available in a real-world scenario) to rank the samples using each feature in turn, calculates the MRR of each ranking and eventually gives us the feature that produces the highest MRR (remember from Section 2, Chapter 2, that MRR is designed exactly for this). Now we can order the documents of each query using this feature's values and label each instance in the order they appear in the ranking. If we find a relevant instance before we reach 16 labeled instances, than we can either stop labeling in the ranked order and pick the remaining instances (up to 16) randomly or keep labeling in the ranked order until we have 16 labeled instances. If a relevant document is not found in the first 16 samples, then we label more documents until we reach a set limit, at which point we give up looking for a relevant instance and move on to the next query.

To evaluate the advantage that Donmez's RR method has in relation to ARLR, we implemented two practical variations called RReal and RRealOracle. The differ-

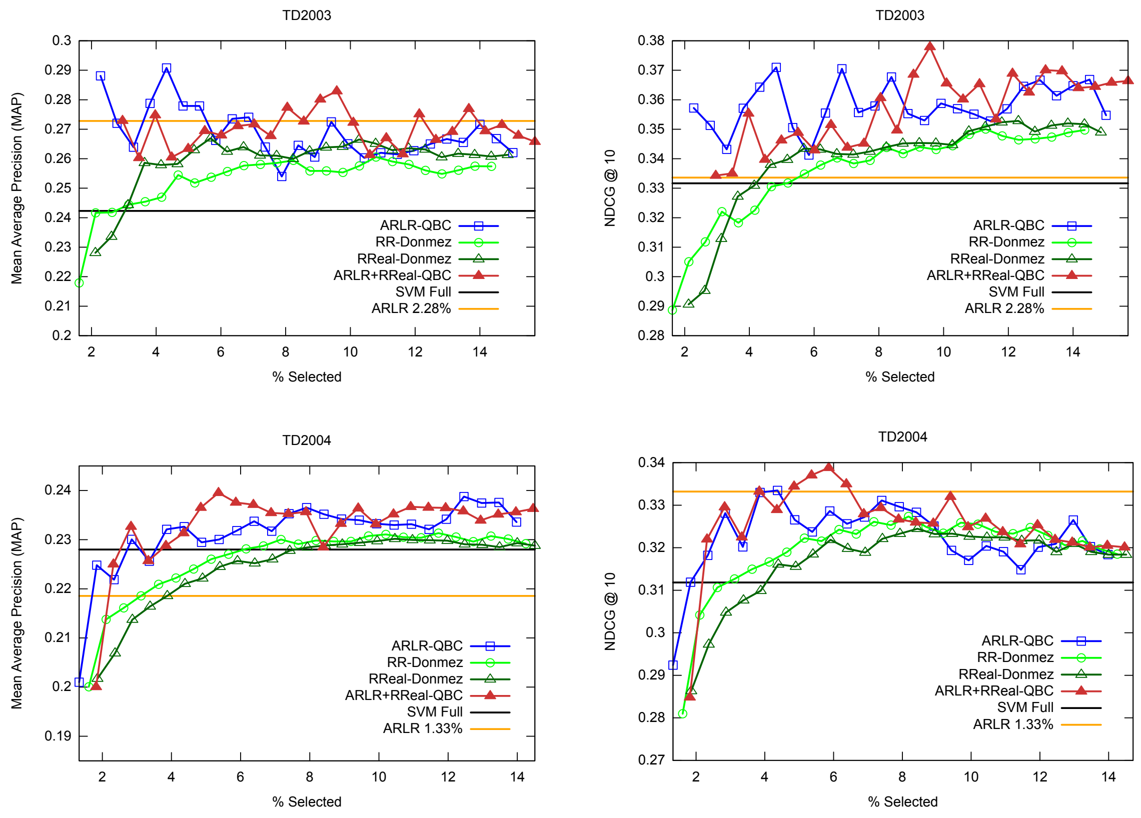

Figure 5.3. TD2003 and TD2004: Comparing RR, RReal and ARLR initial selection strategies; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

ence between these two initial set selection strategies is that in RReal, if a relevant sample is found before 16 are labeled then the remaining are randomly chosen, while in RRealOracle we keep labeling in the ranking order determined by the oracle. In both methods, if a relevant sample is not found in the first 16 instances, then we keep labeling until one is found or until we have labeled 50 documents of that query. RReal is closer to the "ideal" RR, since at least some samples are randomly chosen. The reason we have chosen to also implement RRealOracle is that we want to gauge the influence of the "diversity" provided by the randomly sampled instances, since this is one of the corner stones of ARLR.

First we run Donmez's second stage selection with RReal to measure how the ideal RR may be influencing the results. We call this new baseline RReal-Donmez. Figures 5.3 and 5.4 show the results for this new baseline, comparing them to RR-Donmez and ARLR-QBC for the informational and navigational datasets, respectively. The other new result that appear in the plots (ARLR+RReal-QBC) will be explained in section 4.4.4 below, so just forget about it right now. As we can see in Figure 5.3, RReal-Donmez produces very similar results as RR-Donmez on the TD datasets. Notice that the lines start a little shifted to the right, as RReal selects more documents for the initial sets as a consequence of not finding a relevant instance in the first 16 results for some queries. RReal selects 2.12% of the unlabeled set for TD2003 and 1.86% for TD2004 (RR selects 1.6%). Observe that the y-axis scale has changed in comparison to Figures 5.1 and 5.2.

Figure 5.4 shows the results obtained using RReal-Donmez on the HP and NP datasets. RReal-Donmez is clearly worse than RR-Donmez in HP2004 and NP2003. The results are quite similar for NP2004. In HP2003, although the MAP for the first rounds are slightly better, the NDCG@10 results are much worse. Observe that, compared to RReal-Donmez, ARLR-QBC performs better in the initial rounds for all datasets, with the exception of NP2004.

These results show that RR-Donmez has a small edge over our method in the initial rounds on the navigational datasets mostly because of the synthetic nature of RR. RR is not a practical method like ARLR or RReal and it combines two characteristics that seem to be very important to SVMRank: a diverse set of non-relevant instances picked randomly, coupled with one guaranteed relevant instance per query. RReal, on the other hand, provides almost one relevant instance per query (for some queries it does not find a relevant one before hitting 50 labeled samples) but at the cost of selecting a not-so-diverse set of non relevant examples (since many of them are labeled in the order obtained by the feature ranking method). RReal also selects more instances, as we have seen above. ARLR does not explicitly seek to select relevant instances, although it does tend to select a higher proportion of relevant samples (see Table 4.2, column RS%, for the proportion of relevant instances selected by ARLR; column R% shows the percentage for the full dataset). At the same time, ARLR maximizes diversity in the selected set which seem to help both SVMRank and QBC. But SVMRank depends on relevant instances to produce its pairwise training, which may be an explanation for the not-so-good initial results of ARLR-QBC on the navigational datasets. In the informational datasets, ARLR does select a higher proportion of relevant instances (although not necessarily one per query) simply because there are more relevant instances per query. In TD2004 there are on average 29 relevant instances per query (each query has around 1,000 documents in all datasets), while in the HP and NP datasets, there is on average only 1.3 relevant documents per query (see Table 2.1, column R/Q for the average number of relevant instances per query for each dataset).

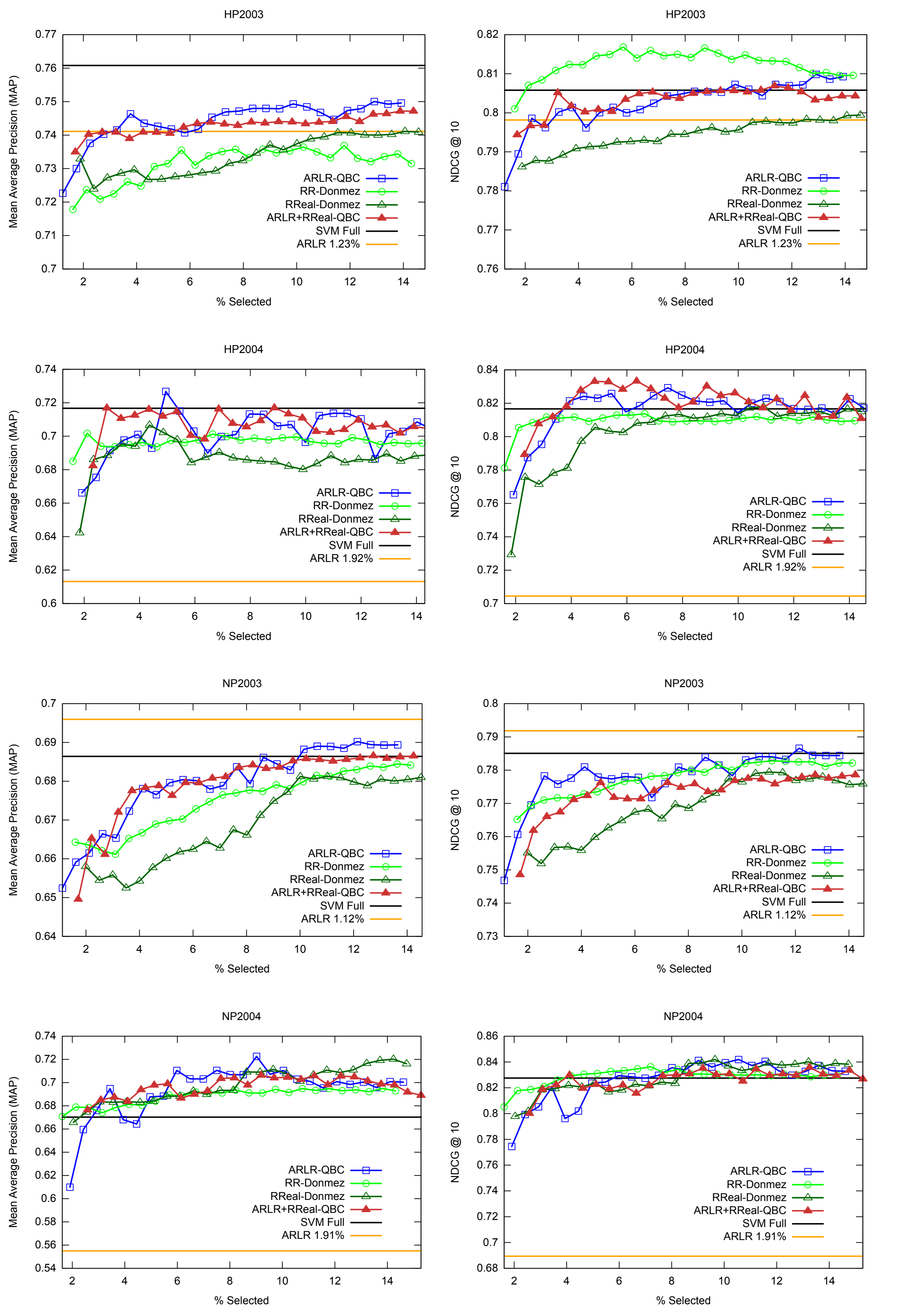

Figure 5.4. HP2003, HP2004, NP2003 and NP2004: Comparing RR, RReal variations and ARLR initial selection strategies; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

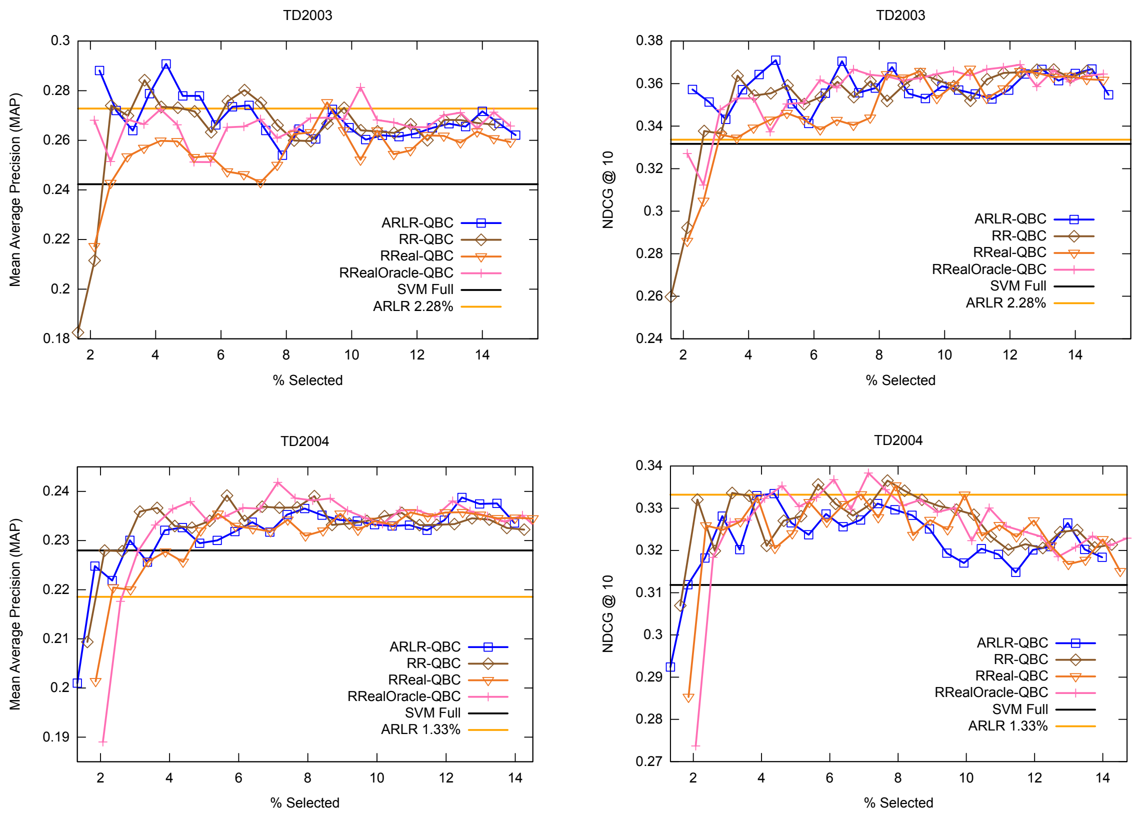

Figure 5.5. TD2003 and TD2004: Comparing RR, RReal, RRealOracle and ARLR initial selection strategies with QBC; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

Summary

Summing up, the initial selection method used by Donmez (RR) is somewhat unrealistic. In an active learning scenario where we have no previous training data, we have to resort to a method such as RReal, which does not yield the same results as RR, specially on the navigational datasets. ARLR-QBC fares better than RReal-Donmez in all datasets with the exception of NP2004 where the results are quite similar.

4.4.2 QBC with different initial selections

To better understand the influence of the initial selection methods on QBC's performance, we ran four new experiments: RR-QBC shows the results of using the "ideal" RR method with QBC second stage selection, RReal-QBC uses the RReal initial set selection method described above and RRealOracle-QBC is the variation of RReal where no instances are randomly selected (i.e. all instances are selected in the order they appear in the oracle-mandated ranking). We also reproduce the results for ARLR-QBC, SVM Full and ARLR $x\%$ to help put these new results in perspective.

Figure 5.5 shows the results for TD2003 and TD2004 and Figure 5.6 the results for the navigational datasets.

In TD2003, RReal-QBC does not perform very well, especially in the initial rounds. RR-QBC starts with a very low MAP and NDCG@10, but quickly picks up, achieving an overall result similar to ARLR-QBC. Surprisingly, RRealOracle-QBC obtains very good results from the start. RRealOracle is very different than RReal in the TD datasets, since by selecting all instances in the oracle-ranked order it actually selects many more relevant instances (see the next section for details on the selection of relevant instances by the various methods). The extra relevant instances seem to help SVMRank, but ARLR-QBC still yields better results, indicating that a combination of relevant instances and diversity is still the best strategy.

For TD2004, the RR-selected set does slightly better than our method in the initial rounds, but when we use RReal, this picture changes: RReal gives us a similar MAP in the initial set (compared to ARLR), but selects more instances (1.86% versus the 1.33% selected by ARLR - notice how the curves start slightly shifted to the right of ARLR-QBC). From around 4% (round 5) on, the results are very similar. Here RRealOracle starts with lower results, but quickly reaches levels similar to ARLR-QBC (from round 2 on). As we noted above, a greater proportion of relevant instances does not necessarily translate into a better model for SVMRank. It seems that we need to strike a balance: SVMRank needs at least one relevant instance per query to be effective, but the diversity of the non relevant instances also helps.

On the navigational datasets (Figure 5.6), the picture is very similar to TD2004: RR-QBC starts very well (beating ARLR-QBC in all datasets), but RReal-QBC does not do so well in the initial rounds (except on NP2004). RRealOracle usually starts with lower results but quickly picks up. The exceptions are NP2004, HP2003, where RRealOracle's results start declining after round 2, and the NDCG@10 results for NP2003.

Summary

Summarizing, ARLR selects very good initial samples on all datasets, but specially on the informational ones (TD2003, TD2004). RR is very good on the navigational datasets, but, as we have discussed, it is not practical. If we use RReal instead, the results for the first few rounds are not as good as those obtained with RR. Overall, using an initial selection method that tries to guarantee at least one relevant instance per query yields good results on the navigational datasets, as SVMRank needs at least one relevant instance per query to produce the pairwise comparisons necessary to train

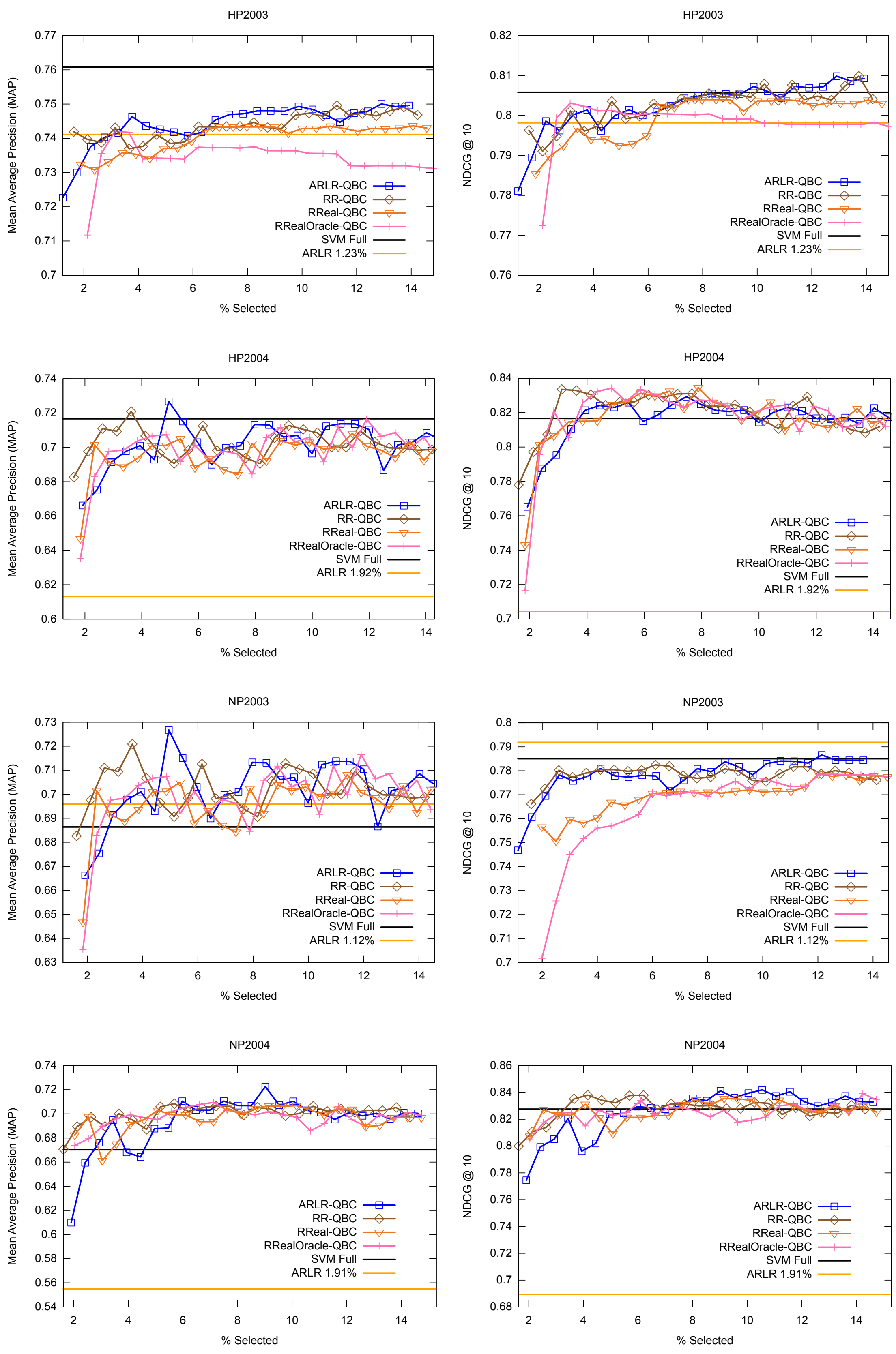

Figure 5.6. HP2003, HP2004, NP2003 and NP2004: Comparing RR, RReal, RRealOracle and ARLR initial selection strategies with QBC; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

the learned model.

4.4.3 Relevant instances selection

To better understand the differences between these methods and datasets we now look into how the selection of relevant instances may be affecting the results. There are two simple metrics that may help us gauge the influence of relevant instances in the results: the proportion of relevant instances in the selected sets and the fraction of all relevant instances present in the unlabeled sets that are selected.

In Figure 5.7, the plot on the left shows, for each method, how the proportion of relevant instances in the selected sets evolve as we run the rounds of the methods. On the y-axis we have the percentage of relevant instances in the selected sets for each round (x-axis). The plot to the right shows the fraction these relevant selected samples represent of the *total* amount of relevant documents present in the unlabeled sets.

We first notice how the informational (first row of Figure 5.7) and navigational datasets (second and third rows) differ in these metrics. For the TD datasets, the proportion of relevant samples in the selected set *grows* in the first few rounds, indicating that both QBC and Donmez are selecting more relevant instances than non-relevant. Then the proportion gently slopes down, although by the 25^{th} round it is still higher (or the same) than in the initial round. Observe from the plot on the right that the initial relevant samples selected represent a minor proportion (3% to 20%) of all relevant samples in the sets, so both round-based methods have room to find more relevant samples. On the HP and NP datasets, on the other hand, all RR-derived methods select almost *all* relevant samples available in the initial round (plots on the right: from 75% to 97%) which means that as the rounds progress the tendency is to select more non relevant samples, gradually reducing the proportion of relevant documents in the selected sets (plot to the left). We also notice that ARLR-QBC perform similarly to the RR-derived methods in the informational datasets, while in the navigational datasets there is a remarked difference.

Let's take a more detailed look at the plots for TD2004: we can see that, although RR starts with over 6.5% of relevant instances in the selected set, both RReal and ARLR perform similarly well (6.1% and 5.8%). These samples represent only 7.5%, 7.2% and 5% of the total number of relevant instances in the unlabeled sets, respectively. This means that QBC and Donmez have room to increase the proportion of relevant samples in the selected sets. We can also see that Donmez has a harder time finding relevant instances. By round 25, from 67% to 69% of all relevant samples in the unlabeled set have been selected by QBC-based methods, while Donmez reaches

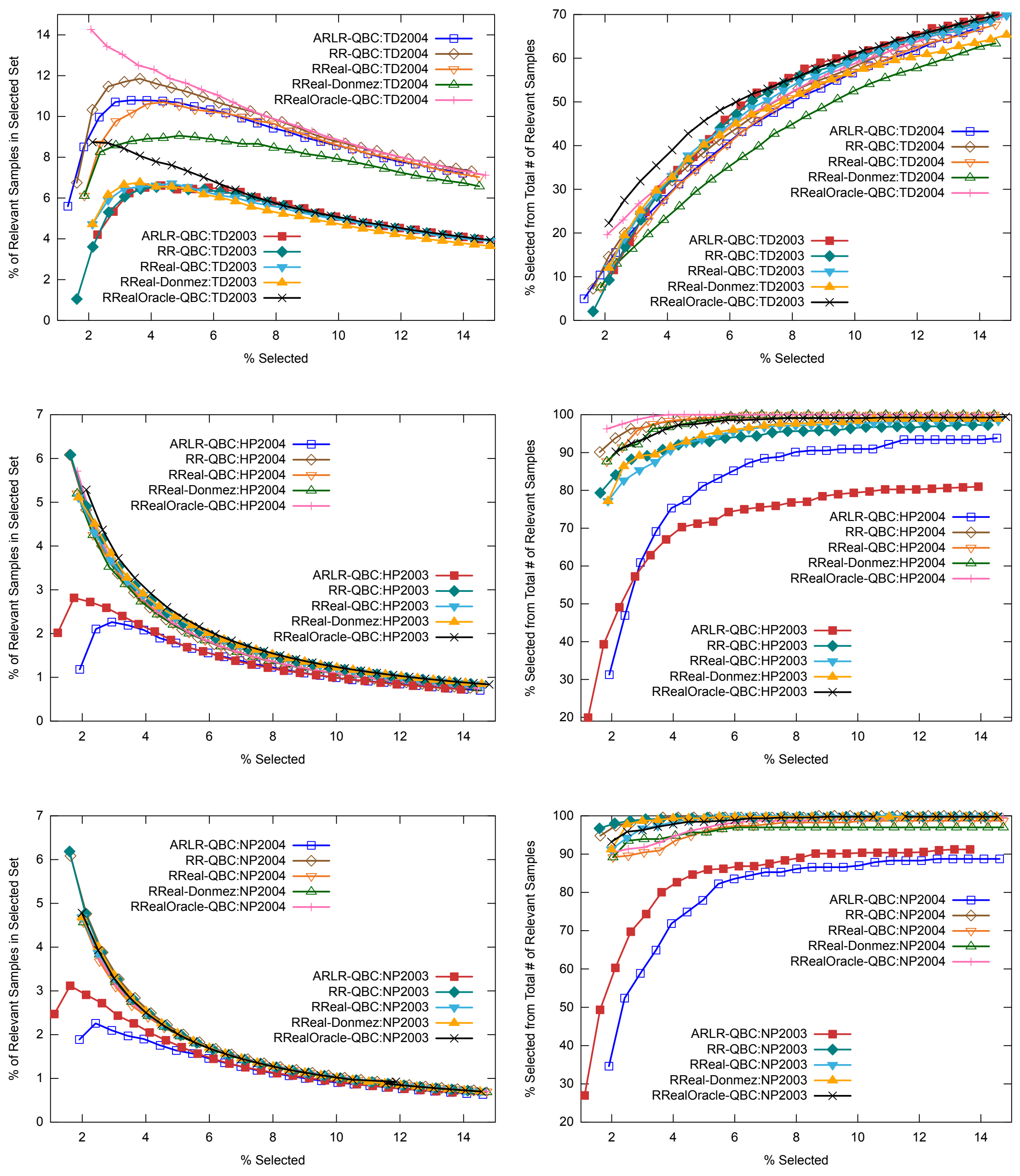

Figure 5.7. All datasets: Proportion of relevant instances in the selected sets (left) and the percentage they represent of the total number of relevant samples in the unlabeled sets (right)

62%. Also notice that the RRealOracle plots on the left are quite different from the rest: it selects a higher proportion (almost twice that of RR-based selection) of relevant samples in the initial round since it selects all instances in the oracle-determined ranking order. The oracle chooses the feature that maximizes the MAP, which means that, on the TD datasets, usually more than one relevant sample will appear in the first 16 instances of the ranked list.

Compare that to the plots for HP2003: RR and RReal initial sets start with 6.2% and 5.2% of relevant instances. This represents 79% and 77% of all relevant samples in the training sets, respectively. In both cases, this proportion reaches 95% by round 10. This means both QBC and Donmez are quickly finding the remaining relevant samples, but they also quickly run out of relevant instances to select. ARLR, on the other hand, starts with around 2% of relevant samples in the selected set (20% of the total) and QBC then finds about 70% of all relevant samples by round 6 (by round 25, this number reaches 80%). Remember that most queries in the navigational datasets have only 1 relevant document; since RR-based selection tries to obtain exactly 1 relevant instance per query, this means that the initial set contains almost all relevant samples in the unlabeled set. Although it seems clear that having at least one relevant sample for each query helps SVMRank, ARLR-QBC is able to select very informative instances that yield very good results with SVMRank, as we can see in Figures 5.3 and 5.4.

Summary

In short, although ARLR does not select nearly as many relevant instances as the other methods, the diversity of the selected samples compensate for the lack of relevant instances, even on the navigational datasets. On the navigational datasets, the RR-based methods select almost all relevant instances available in the unlabeled sets in the initial round, while ARLR selects much smaller proportions. This difference does not reflect directly in the results, indicating that ARLR compensates the lack of relevant samples with more diverse non-relevant documents. This conclusion is somehow corroborated by the better initial results obtained by RR and RReal, in comparison to RRealOracle, in all datasets with the exception of NP2004 and TD2003. All three methods select relevant instances for almost all queries, but only RR and RReal also select some instances randomly.

Furthermore, QBC is very effective at selecting relevant instances when ARLR is used as the initial selection method, as can be seen in Figure 5.7 on the plots to the right. It also selects interesting instances, achieving very good results quickly even when the initial set does not perform so well, as can be seen from the RRealOracle results on almost all datasets with the exception of TD2003 and NP2004.

4.4.4 Extending ARLR with RReal

The RReal results indicate that RR-Donmez obtains an artificial advantage over our method in the initial rounds for some navigational datasets. Although RReal may provide better initial results in this type of datasets, substituting ARLR for RReal in our method usually yields worse results in the first few rounds, especially for the TD datasets, but also for HP2003 and NP2003 (NDCG@10). The strength of ARLR lies in selecting very diverse samples and this strategy helps QBC not only in informational datasets but also in navigational ones such as HP2003. The main advantage of ARLR-QBC is to provide a practical and simple method to select a small training set from scratch that yields effective results. But the RR-based methods tend to perform well in the initial rounds on the navigational datasets. As we have seen in the last section, having more queries with one relevant instance helps SVMRank in the initial rounds.

Maybe we can improve the results by merging the strengths of ARLR and RReal. One way of doing that is to run ARLR and label the initial sets; then we can verify for which queries ARLR did not select at least one relevant sample and run RReal on these queries, labeling more instances in an attempt to find one relevant document. Observe that in this case, this RReal variant stops either when a relevant sample is found or 50 new (non relevant) documents have been labeled (i.e. there are no samples selected randomly). The results for this hybrid initial selection method appear in Figures 5.3 and 5.4 as “ARLR+RReal-QBC”. From the plots we can see that this new method provides slightly better results in the initial rounds in HP2004 and NP2004 which are exactly the datasets where RReal-Donmez held a small advantage over ARLR-QBC. For HP2003 and NP2003 (MAP) the results are roughly the same, while in TD2003 they are a bit worse. For TD2004 the results are comparable from the 2^{nd} round on. This comes at the cost of labeling more instances in the initial round. ARLR+RReal selects 2.96% of the unlabeled instances in TD2003, 1.82% in TD2004, 1.70% in HP2003, 2.32% in HP2004, 1.71% in NP2003 and 2.58% in NP2004. It selects more instances in the initial round than RReal in TD2003, HP2004 and NP2004.

Summary

In summary, it could be a good idea to use a hybrid initial selection method (i.e. ARLR+RReal) on navigational datasets, specially if the labeling budget is very tight, since it yields better initial results.

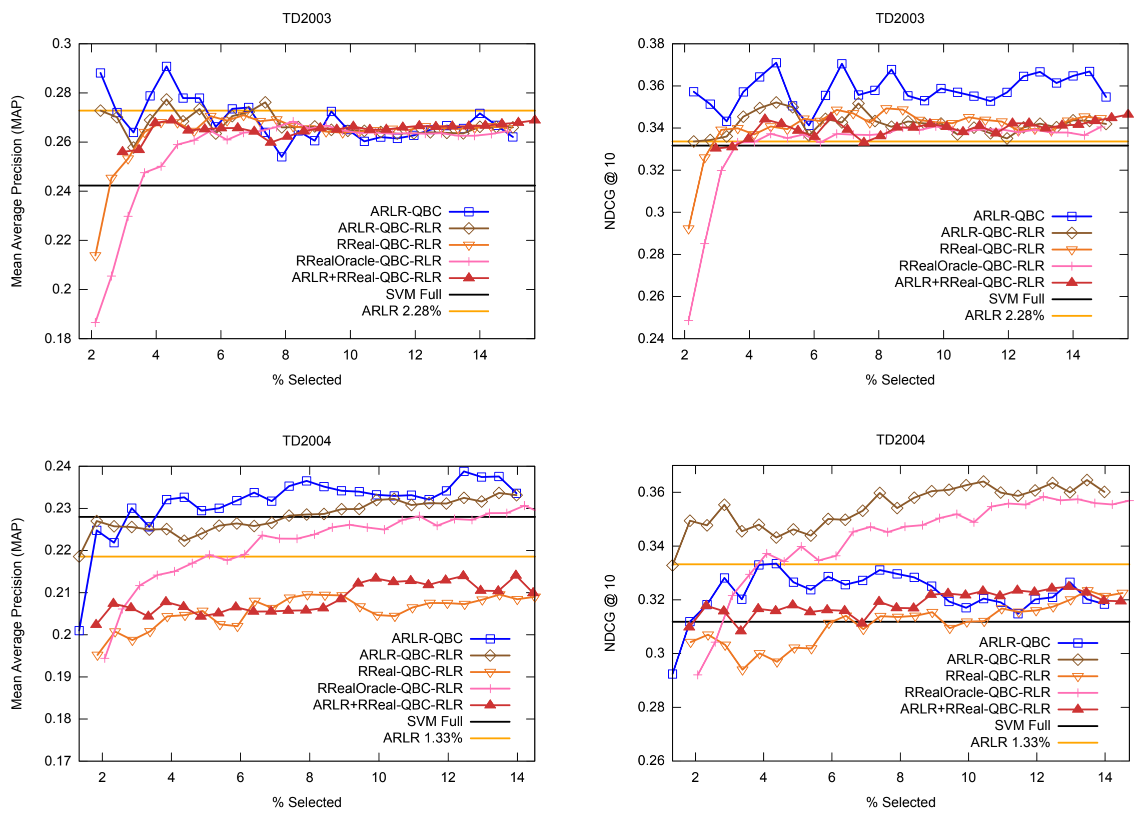

Figure 5.8. TD2003 and TD2004: Ranking using RLR with the selected sets; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

4.5 Using RLR with the selected sets

So far, the results we have presented were obtained by using SVMRank to rank the test sets after the labeled sets were produced using the various combinations of first stage selection (i.e. ARLR, RR, RReal) and second stage, round-based methods (i.e. QBC and Donmez). However, the selected and labeled sets can be used as training by another supervised learning to rank algorithm to rank the test sets.

Figures 5.8 and 5.9 show the results of using RLR to rank the test set after the training sets have been selected and labeled. We run RLR with four of the first stage selection variations and with QBC as the second stage selection method. Namely, we test ARLR-QBC-RLR, RReal-QBC-RLR, RRealOracle-QBC-RLR and ARLR+RReal-QBC-RLR using the exact same selected tests as presented above, but with RLR performing the ranking of the test sets (instead of SVMRank). We also include in the plots the results for ARLR-QBC, SVM Full and ARLR $x\%$ for easier comparison.

The results for the informational datasets (Figure 5.8) show that RLR (i.e. ARLR-QBC-RLR) performs very well with ARLR-QBC selected instances. Although it does not surpass SVMRank (i.e. ARLR-QBC) in TD2003 on either metric, it ob-

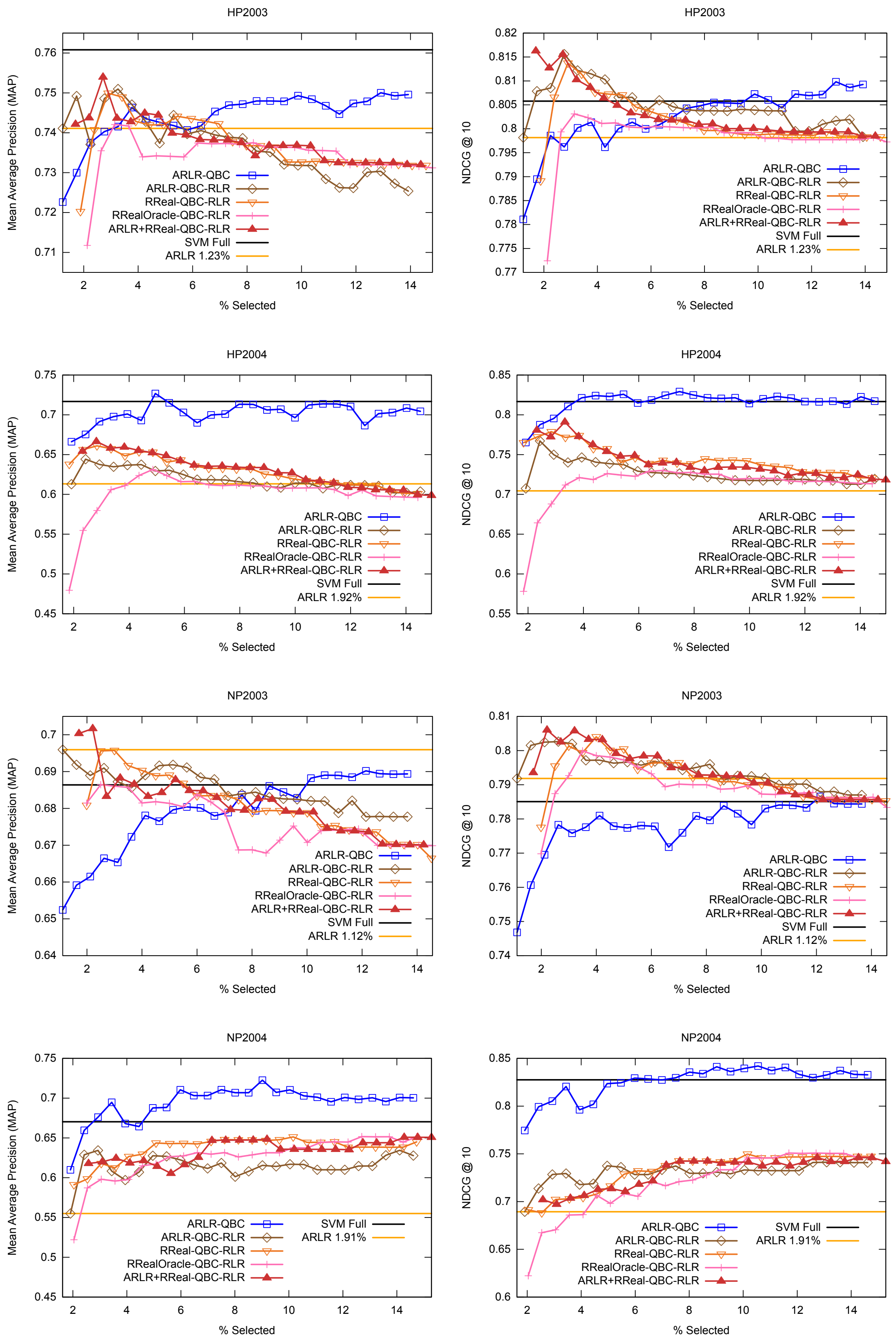

Figure 5.9. HP2003, HP2004, NP2003 and NP2004: Ranking using RLR with the selected sets; MAP (left) and NDCG @ 10 (right) vs. % of samples selected from unlabeled set

tains, nonetheless, very good results, beating SVM Full from the initial to the last round. For TD2004, the MAP results are comparable to ARLR-QBC; the surprise are the NDCG@10 results which not only beat SVMRank, but also the best supervised baseline published by the LETOR producers which is RankBoost with 0.3504 (see Table 5.1). The extra relevant instances selected by ARLR+RReal do not seem to help on the informational datasets; in fact, although it fairs similarly to ARLR-QBC-RLR on TD2003, the resulting MAP is significantly worse for TD2004 (although the NDCG@10 is only slightly lower than ARLR-QBC). For TD2003, RReal and RRealOracle fare similarly, starting with lower results but quickly reaching the same level as the other methods. On TD2004, RRealOracle achieves impressive results, although not reaching the level of ARLR-QBC (both metrics). As we can see from the plots, on TD2003, RLR performs similarly for all selection methods, although RReal and RRealOracle start with lower results in the first few rounds. On TD2004, the picture is very different with two very different selection methods performing quite well (ARLR and RRealOracle), specially on the NDCG@10 results, and the other two (ARLR+RReal and RReal) obtaining lower MAP results.

Figure 5.9 shows the results for the navigational datasets. Here we see a very different trend, with RLR results for all datasets, except NP2004, starting at or quickly reaching its peak results in the first few rounds and then slowly descending as more instances are inserted into the training set. Both ARLR-QBC-RLR and ARLR+RReal-QBC-RLR obtain very similar results on all datasets, with RReal and RRealOracle starting with lower results but quickly picking up. On the datasets where ARLR $x\%$ fares very well (i.e. HP2003 and NP2003), the RLR results for the first few rounds are very good, specially on NP2003 (both metrics), but also on HP2003 (NDCG@10). On HP2004 and NP2004, although the RLR results never reach the same level of ARLR-QBC or SVM Full, they do beat ARLR $x\%$, indicating that QBC selection is very effective for RLR (remember that RLR is one of the committee algorithms). The best results obtained by RLR were on the initial rounds for NP2003 where it beats all LETOR supervised baselines with the exception of RankBoost (MAP 0.7074, NDCG 0.8068).

Summary

Summarizing, the selected sets perform well with RLR, specially ARLR-QBC, which improves the results obtained by RLR with ARLR-only data on all datasets. RLR seems to be more easily affected by the proportion of relevant instances in the selected sets as can be seen from the declining results on HP2003, HP2004 and NP2003.

4.6 Comparing ARLR-QBC to LETOR baselines

In Section 3.3.4 of Chapter 4, we compared ARLR results to some of the published LETOR baselines. Here we compare the results obtained using ARLR-QBC to those obtained by the 12 published baseline results.

In TD2003, using only the initial dataset (i.e. round 0), ARLR-QBC obtains a MAP of 0.2881, surpassing all of the 12 supervised baselines published by the LETOR producers. ARLR-QBC manages to obtain an even higher MAP at round 4 (0.2908) and a peak NDCG of 0.3710 at round 5. In TD2004, the MAP for all rounds (with the exception of rounds 0 and 2) beat 9 of the 12 supervised baselines published by the LETOR producers and the NDCG (with the exception of rounds 0, 1 and 20), beat 8 out of the 12 baselines. If we consider only the peak results (round 22 for the MAP and round 6 for the NDCG), then ARLR-QBC beats 11 of the 12 supervised baselines (loosing only to RankBoost).

In HP2003, ARLR-QBC obtains a MAP of 0.7463 by round 5 (selecting only 3.77% of the unlabeled set), surpassing 5 of these baselines (including SVMRank and RankBoost). The NDCG results are a little worse, beating only 3 of these baselines (Regression, SVMMAP and FRank) by round 5 (with a value of 0.8013). The results are much better for HP2004, where by round 6 (4.95% selected) the MAP obtained (0.7268) beats all 12 published supervised baselines. The NDCG@10 for round 5 (0.8241) is better than that for all baselines with the exception of AdaRank-MAP. On NP2003, although ARLR-QBC never reaches the same performance level of SVM Full, by round 22 (12.14% selected) the MAP of 0.6902 is still better that of 9 of the 12 supervised baselines. The NDCG fares a little worse, surpassing only 4 of these baselines by round 3. Our method also obtains outstanding results on NP2004 beating all supervised baselines by round 3 with a MAP of 0.6948 (and reaching a peak of 0.7226 in round 14, which represents a gain of 5.24% over the best baseline, Regression-Reg). The NDCG@10 obtained at round 3, 0.8205, also surpasses all 12 baselines, achieving a peak value of 0.8419 in round 17 (a gain of 3.58% over the best baseline, ListNet).

Figure 5.10 shows the comparison of the MAP obtained by ARLR-QBC, ARLR and three LETOR baselines: SVMRank, RankBoost and FRank. The ARLR-QBC results shown are the peak values obtained in rounds 0 to 7. As we can see in the Figure, ARLR-QBC obtains better results than ARLR in all datasets, with the exception of NP2003. ARLR-QBC also beats the chosen LETOR baselines in TD2003, HP2003, HP2004 and NP2004. Notice that ARLR-QBC obtains specially good results on the datasets where ARLR did worse: HP2004 and NP2004.

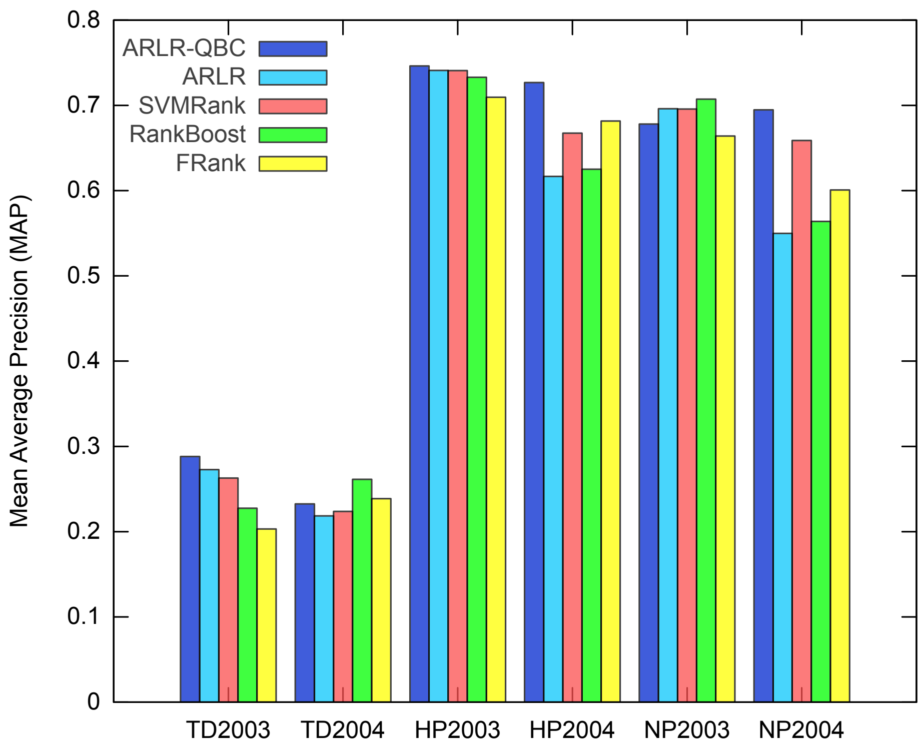

Figure 5.10. Comparison of ARLR-QBC, ARLR and three LETOR baselines: Best MAP obtained in rounds 0 to 7

4.7 Gains obtained over RReal-Donmez and SVMRank

Table 5.1 summarizes the gains obtained by our method compared to RReal-Donmez and the SVMRank results published by the LETOR producers. We calculate the gains for each metric separately: MAP (a) and NDCG@10 (b). We also separate the the calculations into two partitions: the average and maximum gains achieved in rounds 0 to 7 (columns AG7% and MG7%, respectively) and the overall (i.e. considering all rounds) average and maximum gains (AG% and MG%). The reason for doing this partitioning is that, although we run our method for 26 rounds, we believe that in most real-world scenarios only a few rounds should be run. One of the advantages of an active learning method is exactly to introduce less "noise" into the selected training sets, thus allowing for the supervised L2R method to obtain better effectiveness. This insight is corroborated by the gains obtained in all datasets (except NP2003) by our method over supervised baselines that use the complete training sets (i.e. SVMRank and the other published LETOR baselines) and by the stable or falling results that we see in many datasets as the rounds progress. By providing the average and max gains obtained at the first 8 rounds (0-7) we want to show that our method converges faster to good results as compared to RReal-Donmez and also that it obtains competitive

Table 5.1. Gains obtained by ARLR-QBC over RReal-Donmez and SVMRank: MAP (a) and NDCG@10 (b). Average Gain for rounds 0-7 (AG7%), Max Gain for rounds 0-7 (MG7%), Average Gain for all rounds (AG%) and Max Gain for all rounds (MG%). The number in parentheses indicate in which round the maximum gain was obtained. Values in *italic* indicate negative gains, values in **bold** indicate a gain over 5% and values in ***italic and bold*** a gain over 10%

(a)

	ARLR-QBC vs. RReal-Donmez				**ARLR-QBC vs. SVMRank**			
MAP	AG7%	MG7%	AG%	MG%	AG7%	MG7%	AG%	MG%
TD2003	***10.54***	***26.32*** (0)	3.93	***26.32*** (0)	**5.39**	***10.65*** (4)	2.27	***10.65*** (4)
TD2004	**6.47**	***11.43*** (1)	3.65	***11.43*** (1)	0.44	3.99 (6)	3.38	**6.73** (22)
HP2003	1.69	2.31 (6)	1.55	2.31 (6)	*-0.37*	0.75 (5)	0.44	1.25 (23)
HP2004	0.99	3.71 (0)	2.20	4.30 (19)	4.24	**8.88** (6)	**5.16**	**8.88** (6)
NP2003	2.22	3.64 (6)	1.89	3.64 (6)	*-4.20*	*-2.53* (6)	*-2.38*	*-0.79* (22)
NP2004	*-1.80*	1.71 (3)	*-0.69*	3.22 (8)	1.49	**5.46** (3)	**5.24**	**9.68** (14)

(b)

	ARLR-QBC vs. RReal-Donmez				**ARLR-QBC vs. SVMRank**			
NDCG	AG7%	MG7%	AG%	MG%	AG7%	MG7%	AG%	MG%
TD2003	***10.39***	***22.92*** (0)	**5.68**	***22.92*** (0)	2.42	**7.19** (5)	3.47	**7.19** (5)
TD2004	**6.31**	**8.93** (1)	2.17	**8.93** (1)	4.12	**8.34** (6)	4.71	**8.34** (6)
HP2003	1.04	1.40 (4)	1.15	1.46 (23)	*-1.52*	*-0.78* (5)	*-0.67*	0.26 (23)
HP2004	3.40	**5.15** (4)	1.66	**5.15** (4)	4.93	**7.43** (7)	**6.12**	**7.88** (11)
NP2003	2.73	3.51 (3)	1.48	3.51 (3)	*-3.67*	*-2.42* (6)	*-2.79*	*-1.71* (22)
NP2004	*-1.15*	0.93 (7)	*-0.28*	1.49 (12)	*-0.06*	2.26 (7)	2.40	4.43 (17)

results selecting less than 6% of the original training sets when compared to a strong supervised method using the complete sets (i.e. SVMRank).

Average and Maximum Gains over RReal-Donmez: From Table 5.1 we can see that ARLR-QBC obtains positive gains over RReal-Donmez on all datasets with the exception of NP2004. The improvement is more impressive on the informational datasets, where ARLR-QBC has average results on rounds 0-7 that are over 10% better than RReal-Donmez on TD2003 (both metrics) and over 6% better on TD2004 (both metrics). The overall average gains are also good, reaching over 5% for the NDCG on TD2003. The results for the navigational datasets are more modest, but still quit reasonable, with the average gain on rounds 0-7 reaching 3.4% on HP2004 (NDCG). Observe from the MG% column that the maximum gain is very often obtained in the initial rounds.

Average and Maximum Gains over SVMRank: From the average gain obtained over SVMRank in the first 8 rounds (column AG7% to the right), we can see that ARLR-QBC surpasses this strong supervised baseline in 4 out of 6 datasets for the MAP and half of the datasets for the NDCG@10. This means that our method is able to surpass this strong supervised baseline (which uses 100% of the training sets) while selecting and labeling less than 6% of the original training sets. Moreover, the overall average gain (AG% to the right) is positive in 5 of the 6 datasets for the MAP and 4 for the NDCG. The AG reaches over 5% for the MAP on HP2004 and NP2004, over 6% for the NDCG on HP2004 and over 4% for TD2004.

Paired Student's t-test

We have also performed a paired difference t-test using the MAP and NDCG differences for all rounds of our method in comparison to RReal-Donmez's results. The test shows that our method is significantly better with 95% confidence level in all datasets but NP2004 for the NDCG@10 and on 4 datasets for the MAP (the exceptions are NP2003 and NP2004).

5 Summary

The proposed active learning algorithm provides an effective and practical method to reduce the cost of creating training sets for use with L2R techniques. The method is practical because, as is shown by the results, it selects very effective training sets from very diverse datasets and provides consistent quality for small or bigger labeling budgets. Furthermore, the method does not require the availability of an initial seed training set. ARLR is used to bootstrap the labeling process and provides the second stage selection strategy (QBC) an effective initial set. QBC can then be used for as many rounds as desirable to select more instances.

By using ARLR in conjunction with QBC, we were able to obtain results that are as good as or better than established supervised baselines but selecting only a small fraction of the original training sets. If we look back at Tables 4.5 and 5.1 from Chapter 4, we can see that in the datasets where ARLR did not perform so well (i.e. HP2004 and NP2004), ARLR-QBC achieves much better results with just a few rounds. In HP2004, ARLR-QBC beats the chosen baselines by round 3 (MAP) and by round 2 (NDCG). In NP2004, this happens by round 2 (MAP) and by round 6 (NDCG). In TD2004, our method does not beat the best baseline (RankBoost), but comes very close, specially on the NDCG obtained at round 6.

The sets selected by ARLR-QBC also seem to be useful to different L2R algorithms. The results obtained by using the selected sets as training for RLR show that, although it does not reach SVM Full's level in some datasets (specially HP2004 and NP2004), it does improve on ARLR's results for all datasets, and even beats all published LETOR baselines in some cases (NDCG on TD2004 and NP2003 and MAP on NP2003).

Observe that although we run our method for 25 rounds (selecting almost 15% of the original training sets), very good results are obtained in all datasets in the few initial rounds. On most datasets, 6 or 7 rounds suffice to achieve supervised-level performance. Thus, by labeling only about 5% of the original training sets we are able to achieve a high quality and effective ranking function.

Chapter 6

Conclusions

In this work we have proposed two complementary active learning to rank methods that are both practical and effective. The first one, ARLR, is an associative rule-based method that tries to maximize the diversity of the actively selected set. Extensive experimentation using the LETOR 3.0 web datasets showed that this method is very effective, selecting a very small fraction of the unlabeled set. Despite its effectiveness, the method is not easily extended to allow for more instances to be selected, even if there is labeling budget available or if the desired quality level has not been achieved. To address this shortcoming, we proposed a QBC-based second stage that allows for as many more instances as desired to be chosen. Together, these methods complement each other, facilitating a flexible and highly effective active learning method. The experiments show that this hybrid, two-stage method beats or achieves similar results as state-of-the-art supervised algorithms that use the complete training sets, whilst selecting for labeling a fraction of the unlabeled instances.

1 Future Work

Several questions remain to be answered regarding the proposed method and can be explored in future work.

- We have chosen to run our experiments on six LETOR 3.0 datasets for the simple reason that there are readily available supervised baselines' results. Now that we have shown the effectiveness of the proposed methods against these baselines, it would be interesting to run some experiments using other L2R datasets such as the MSLR-WEBxK[1] and the Yahoo L2R challenge datasets[2].

- One issue that was touched upon in Chapter 5 is the extension of ARLR-QBC so that it can do both query-level and document-level selection simultaneously. The LETOR 3.0 web datasets are not very adequate for query-level selection, since each dataset contains a reduced number of queries (see Table 2.1). Since ARLR-QBC selects a predefined number of documents per query, it is important to be able to do query-level selection on datasets that have large numbers of queries (such as the ones cited above). A simple way of implementing a two-level selection process would be to first select a number of "interesting" queries using a procedure similar to the QBC second-stage of our method, but using Kendall's τ metric as a measure of how much different ranking algorithms disagree about how to rank the queries. Then, ARLR-QBC could be used to select interesting documents from these queries.

- Related to the query-level selection issue described above is the adjustment of the number of documents selected per query, per round. We have decided to select 5 documents per query, per round only to make the comparison to Donmez's method easier. As we have seen, this number accrues to selecting 0.5% of the original datasets per round. It would be interesting to know how changing this number affects the results. This is even more important when query-level selection is used, since we need to determine how many queries should be selected and then how many documents per query. This "batch-mode" active learning is useful for a practical reason, since a batch of documents can be labeled at each round, instead of having to label the documents iteratively, one-by-one.

- Another analysis that has not been developed in the current work is how the selected sets perform with the third algorithm in the committee (namely, RankBoost). As we have seen, ARLR-QBC selects training sets that are effective for both SVMRank and RLR, although each supervised algorithm fares better in some datasets than the other. Is this true also for RankBoost? Another related inquiry is what would be the effect of removing one of the algorithms from the committee? Similarly, what happens if more algorithms are added to the committee?

- ARLR can be modified in several ways which we have not tested. For instance, to further expand the size of selected sets, we could use a semi-supervised approach

[1]http://research.microsoft.com/en-us/projects/mslr/download.aspx
[2]http://learningtorankchallenge.yahoo.com/

and add those unlabeled instances that are classified with high probability by RLR. In this scenario, the unlabeled set would be classified using the labeled set selected by ARLR and those instances which are classified as being relevant or non-relevant with high probability added to the labeled set with the predicted label.

- Another way to increase the selected sets which we have not investigated thoroughly is to use ARLR in a round-based fashion. As we have seen, ARLR naturally stops selecting new instances when it selects a sample that has already been selected and labeled. Another way to select more instances is to wait for ARLR to converge and then remove the selected samples from the unlabeled set, starting a new selection round. From the few tests we performed, this process tends to quickly converge to the point where only one new sample is selected per round. Although we did not obtain good results from these tests on the LETOR datasets, this strategy could be useful in certain situations or datasets. For instance, a similar approach was successfully used for a classification task on a set of author name disambiguation datasets. The resulting paper (Ferreira et al. [2012]) was accepted at the Joint Conference on Digital Libraries 2012 and is pending publication.

- The partitioning process used in ARLR may also be modified to use a different criterion in separating the feature groups. We have used the χ^2 of each feature in relation to the others in order to try to identify the features that are similarly "informative" so we could separate them into different partitions and maximize each partition's "quality". Another more direct route could be to use Principal Component Analysis (PCA) to determine how "similar" each feature is to others. Observe that the features used in L2R (such as those described for the LETOR datasets) usually have a lot of redundancy. For example, feature 21 is the BM25 of the document body, while feature 25 is the BM25 for the whole document (including title, URL, anchor and body). In general, features like these will have similar values and should be put into different partitions to minimize intra-partition redundancy. PCA could prove to be a better method for doing this separation.

- Another important aspect of ARLR (and RLR) is the discretization of the datasets, as we have seen. Although TUBE unsupervised discretization obtains very good results, even compared to a supervised method such as the one by Fayyad and Irani [1993], other unsupervised methods could be used. There are some unsupervised methods proposed in the literature, such as the PCA-based Mehta et al. [2004] or Ludl and Widmer [2000] which try to discretize values while preserving the relationship among features. It would be interesting to implement these methods to verify if it is possible to improve ARLR and RLR results with better discretization techniques.

Bibliography

Abe, N. and Mamitsuka, H. (1998). Query learning strategies using boosting and bagging. In *Proceedings of the 15th International Conference on Machine Learning*, ICML '98, pages 1–9, Madison, Wisconsin, USA. Morgan Kaufmann Publishers Inc.

Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, pages 207–216, Washington, DC, USA. ACM Press.

Almeida, H. M. d., Gonçalves, M. A., Cristo, M., and Calado, P. (2007). A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 399–406, Amsterdam, The Netherlands. ACM Press.

Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern information retrieval: the concepts and technology behind search.* Addison Wesley.

Baum, E. B. (1991). Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks / a Publication of the IEEE Neural Networks Council*, 2(1):5–19.

Burges, C., Shaked, T., Renshaw, E., Lazier, A., Deeds, M., Hamilton, N., and Hullender, G. (2005). Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML '05, pages 89–96, Bonn, Germany. ACM Press.

Cai, P., Gao, W., Zhou, A., and Wong, K. (2011). Relevant knowledge helps in choosing right teacher. In *Proceedings of the 34th international ACM SIGIR conference on research and development in information retrieval*, SIGIR '11, pages 115–124, Beijing, China. ACM Press.

Cao, Y., Xu, J., Liu, T., Li, H., Huang, Y., and Hon, H. (2006). Adapting ranking SVM to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 186–193, Seattle, Washington, USA. ACM Press.

Crammer, K. and Singer, Y. (2001). Pranking with ranking. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, NIPS '01, pages 641–647. MIT Press.

Dagan, I. and Engelson, S. (1995). Committee-Based sampling for training probabilistic classifiers. In *Proceedings of the 12th International Conference on Machine learning*, ICML '95, pages 150–157, Tahoe City, California, USA. Morgan Kaufmann Publishers Inc.

Donmez, P. and Carbonell, J. G. (2008). Optimizing estimated loss reduction for active sampling in rank learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, pages 248–255, Helsinki, Finland. ACM Press.

Donmez, P. and Carbonell, J. G. (2009). Active sampling for rank learning via optimizing the area under the ROC curve. In *Proceedings of the 31th European Conference on IR Research on Advances in Information Retrieval*, pages 78–89, Toulouse, France. Springer-Verlag.

Donmez, P., Carbonell, J. G., and Bennett, P. N. (2007). Dual strategy active learning. In *Proceedings of the 18th European conference on Machine Learning*, ECML PKDD '07, pages 116–127, Berlin, Heidelberg. Springer-Verlag.

Fayyad, U. M. and Irani, K. B. (1993). Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1027.

Ferreira, A., Silva, R., Gonçalves, M., Veloso, A., and Laender, A. (2012). Active associative sampling for author name disambiguation. In *Joint Conference on Digital Libraries*, JCDL '12, pages 175–184, Washington, DC, USA. ACM Press.

Freund, Y., Iyer, R., Schapire, R. E., and Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Geng, X., Qin, T., Liu, T., Cheng, X., and Li, H. (2011). Selecting optimal training data for learning to rank. *Information Processing and Management*, 47(5):730–741.

Granka, L. A., Joachims, T., and Gay, G. (2004). Eye-tracking analysis of user behavior in WWW search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 478–479, New York, NY, USA. ACM Press.

Guo, Y. and Schuurmans, D. (2008). Discriminative batch mode active learning. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems 20*, pages 593–600. MIT Press, Cambridge, MA.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, SIGKDD '02, pages 133–142. ACM Press.

Kleinberg, J. M. (1999). Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604–632.

Lewis, D. D. and Gale, W. A. (1994). A sequential algorithm for training text classifiers. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 3–12, Dublin, Ireland. Springer-Verlag New York, Inc.

Li, P., Burges, C. J. C., and Wu, Q. (2007). Mcrank: Learning to rank using multiple classification and gradient boosting. In Platt, J. C., Koller, D., Singer, Y., and Roweis, S. T., editors, *Advances in Neural Information Processing Systems*, NIPS '07. Curran Associates, Inc.

Liu, T. (2009). Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331.

Long, B., Chapelle, O., Zhang, Y., Chang, Y., Zheng, Z., and Tseng, B. (2010). Active learning for ranking through expected loss optimization. In *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, pages 267–274, Geneva, Switzerland. ACM Press.

Ludl, M. and Widmer, G. (2000). Relative unsupervised discretization for association rule mining. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '00, pages 148–158, Lyon, France. Springer-Verlag.

Mccallum, A. K. (1998). Employing EM in pool-based active learning for text classification. In *Proceedings of the 15th International Conference on Machine Learning*, ICML '98, pages 350–358, Madison, Wisconsin, USA. Morgan Kaufmann Publishers Inc.

Mehta, S., Parthasarathy, S., and Yang, H. (2004). Correlation preserving discretization. In *Proceedings of the 4th IEEE International Conference on Data Mining*, ICDM '04, pages 479–482, Brighton, UK. IEEE Computer Society.

Nguyen, H. T. and Smeulders, A. (2004). Active learning using pre-clustering. In *Proceedings of the 21st International Conference on Machine learning*, ICML '04, pages 623–630, Banff, Alberta, Canada. ACM Press.

Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The PageRank citation ranking: Bringing order to the web. Technical report.

Qin, T., Liu, T., and Li, H. (2010a). A general approximation framework for direct optimization of information retrieval measures. *Information Retrieval*, 13(4):375–397.

Qin, T., Liu, T., Xu, J., and Li, H. (2010b). LETOR: a benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13:346–374.

Qin, T., Liu, T., Zhang, X., Chen, Z., and Ma, W. (2005). A study of relevance propagation for web search. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '05, pages 408–415, Salvador, BA, Brazil. ACM Press.

Radlinski, F. and Joachims, T. (2007). Active exploration for learning rankings from clickthrough data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, SIGKDD '07, pages 570–579, San Jose, California, USA. ACM Press.

Robertson, S. and Zaragoza, H. (2007). On rank-based effectiveness measures and optimization. *Information Retrieval*, 10(3):321–339.

Schapire, R. E. (1989). The strength of weak learnability. In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, pages 28–33, Washington, DC, USA. IEEE Computer Society.

Scheffer, T., Decomain, C., and Wrobel, S. (2001). Active hidden markov models for information extraction. In *Proceedings of the 4th International Conference on Advances in Intelligent Data Analysis*, IDA '01, pages 309–318, London, UK. Springer-Verlag.

Schmidberger, G. and Frank, E. (2005). Unsupervised discretization using tree-based density estimation. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, PKDD '05, pages 240–251, Berlin, Heidelberg. Springer-Verlag.

Schohn, G. and Cohn, D. (2000). Less is more: Active learning with support vector machines. In *Proceedings of the 17th International Conference on Machine Learning*, ICML '00, pages 839–846, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Settles, B. (2009). Active learning literature survey. Computer Sciences Technical Report 1648, University of Wisconsin-Madison.

Settles, B. and Craven, M. (2008). An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 1070–1079, Stroudsburg, PA, USA. Association for Computational Linguistics.

Settles, B., Craven, M., and Ray, S. (2008). Multiple-instance active learning. In *Advances in Neural Information Processing Systems*, volume 20, pages 1289–1296. MIT Press.

Seung, H. S., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, COLT '92, pages 287–294, Pittsburgh, PA, USA. ACM Press.

Shakery, A. and Zhai, C. (2003). Relevance propagation for topic distillation UIUC TREC-2003 web track experiments. In *Proceedings of TREC*, pages 673–677.

Shashua, A. and Levin, A. (2003). Ranking with large margin principle: Two approaches. In S. Becker, S. T. and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 937–944. MIT Press, Cambridge, MA.

Silva, R., Gonçalves, M. A., and Veloso, A. (2011). Rule-based active sampling for learning to rank. In *Proceedings of the 2011 European Conference on Machine Learning and Knowledge Discovery in Databases*, ECML PKDD '11, pages 240–255, Athens, Greece. Springer-Verlag.

Steck, H. (2007). Hinge rank loss and the area under the ROC curve. In *Proceedings of the 18th European Conference on Machine Learning*, ECML '07, pages 347–358, Warsaw, Poland. Springer-Verlag.

Taylor, M., Guiver, J., Robertson, S., and Minka, T. (2008). SoftRank: optimizing non-smooth rank metrics. In *Proceedings of the International Conference on Web Search and Web Data Mining*, WSDM '08, pages 77–86, Stanford, CA, USA. ACM Press.

Tian, A. and Lease, M. (2011). Active learning to maximize accuracy vs. effort in interactive information retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pages 145–154, Beijing, China. ACM Press.

Tong, S. and Koller, D. (2002). Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research*, 2:45–66.

Tsai, M., Liu, T., Qin, T., Chen, H., and Ma, W. (2007). FRank: a ranking method with fidelity loss. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, page 383–390, Amsterdam, The Netherlands. ACM Press.

Vapnik, V. N. (1995). *The nature of statistical learning theory.* Springer-Verlag New York, Inc., New York, NY, USA.

Veloso, A. and Meira, Jr., W. (2011). *Demand-Driven Associative Classification.* Springer Publishing Company, Incorporated, 1st edition.

Veloso, A. A., Almeida, H. M., Gonçalves, M. A., and Meira, Jr., W. (2008). Learning to rank at query-time using association rules. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 267–274, Singapore, Singapore. ACM Press.

Xu, J. and Li, H. (2007). AdaRank: a boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 391–398, Amsterdam, The Netherlands. ACM Press.

Xu, J., Liu, T., Lu, M., Li, H., and Ma, W. (2008). Directly optimizing evaluation measures in learning to rank. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 107–114, Singapore, Singapore. ACM Press.

Yu, H. (2005). SVM selective sampling for ranking with application to data retrieval. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, SIGKDD '05, pages 354–363, Chicago, Illinois, USA. ACM Press.

Yue, Y., Finley, T., Radlinski, F., and Joachims, T. (2007). A support vector method for optimizing average precision. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '07, pages 271–278, Amsterdam, The Netherlands. ACM Press.

Zhai, C. and Lafferty, J. (2004). A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214.