RESULT MERGING FOR EFFICIENT DISTRIBUTED INFORMATION RETRIEVAL

By Kien-Tsoi T.E. Tjin-Kam-Jet

Master's thesis
University of Twente
March 2009

Graduation committee:

Chairman:	Dr. ir. Djoerd Hiemstra
1st coordinator:	Pavel V. Serdyukov, Msc.
2nd coordinator:	Almer S. Tigelaar, Msc.

University of Twente
Enschede - The Netherlands

Abstract

Centralized Web search has difficulties with crawling and indexing the Visible Web. The Invisible Web is estimated to contain much more content, and this content is even more difficult to crawl.
Metasearch, a form of distributed search, is a possible solution. However, a major problem is how to merge the results from several search engines into a single result list. We train two types of Support Vector Machines (SVMs): a regression model and preference classification model. Round Robin (RR) is used as our merging baseline. We varied the number of search engines being merged, the selection policy, and the document collection size of the engines. Our findings show that RR is the fastest method and that, in a few cases, it performs as well as regression-SVM. Both SVM methods are much slower and, judging by performance, regression-SVM is the best of all three methods. The choice of which method to use depends strongly on the usage scenario. In most cases, we recommend using regression-SVM.

Preface

During my Final Project, I was still on the board of the classical student choir of the Twente University. My last project as a member of the board involved co-organizing a one-day event where everybody could rehearse, relax and have fun, and give a public performance on stage in the evening. This project received the Student Union Culture prize, which is a very nice way to end my membership of the board, and my time as a student.

While looking for a suitable Final Project, I was leaning towards a project that involved Database technology and Natural Language Processing. A logical step was to discuss my plans with my study advisor, Djoerd Hiemstra. I found his suggestion for doing a Final Project on distributed Web search very appealing and so he became my supervisor.

Soon, it became apparent that I should pursue a different direction in the name of Machine Learning, which I first had to become more acquainted with. This field is very interesting, although it would not have been that interesting were it not for the discussions I had with a fellow student, Edwin de Jong.

Working on this research project was very inspiring, not just because of the topic, but also because of the open and friendly research environment.

I would like to thank my supervisors for their guidance and valuable input in my research. I would also like to thank Ander de Keijzer, for providing me with an account on his fast machine. I would also like to express my gratitude to everyone who gave me advice and useful comments, especially, Robin Aly, my fellow lab rats, and Wolter Siemons. Special thanks to my girlfriend, who supported me throughout this project.

Enschede, March 2009 *Kien-Tsoi Tjin-Kam-Jet*

Contents

Chapter 1

Introduction

Finding relevant information on the Web is becoming more important and increasingly more challenging. This chapter exposes the problems with the current state of Web search and introduces a potential solution.

1.1 Motivation

1.1.1 Search Aspects

Every day millions of people all over the world use search engines. A search engine can only provide relevant results if it already has some information about the Web. It obtains this information by downloading Web content. Then, it orders this information, for instance, by means of an inverted file [2], to facilitate quick retrieval of relevant results. Finally, users must somehow be allowed to search within this knowledge. These search aspects are typically referred to as: 1) crawling (a structured way of collecting Web content), 2) indexing (ordering the content for efficient and quick retrieval), and 3) searching.

1.1.2 Web Search Issues

Dominant search engines such as Google, Yahoo!, and Live Search, control all search aspects and store their index in a centralized manner. Even though they use multiple machines to crawl, index, and search the Web, these search engines are still called centralized because; first, they control all search aspects (from one location); second, their machines have complete access to their crawled Web statistics, allowing them to build a true global (centralized) index, as opposed to building many (decentralized) local indices, one for each machine that crawled a piece of the Web. Limitations regarding the centralized index paradigm will be discussed next.

The Web is expected to grow exponentially [38] and already in 1999 it was estimated that no search engine indexed more than 16% of the Visible Web [22]. The Visible Web is a collection of crawlable pages: those pages that can be reached by simply following hyperlinks. The Invisible Web, or Deep Web, is the collection of non-crawlable pages. However, the Deep Web mainly refers to pages that reside behind HTML-forms and that are created dynamically. The size of the Deep Web is estimated to be orders of magnitude (as much as 550 times) larger than the Visible Web [7, 23]. Accessing the Deep Web requires filling in and submitting HTML-forms. Most search engines lack the ability to automatically and adequately fill in and submit these forms, thereby missing possibly relevant pages. In addition, crawlers can be denied access by large websites (e.g., because the website has its own search engine), thereby excluding significant amounts of useful information to centralized search engines [29]. Finally, maintaining and updating centralized indices is not trivial [3].

1.1.3 Metasearch

Metasearch, a form of distributed Web search, potentially solves the problems outlined in the previous section. A metasearch system contains multiple search engines and at least one search broker. Each search engine indexes a distinct part of the Web whereas the broker mediates between the user and these different search engines. At query time, the user sends a query to the broker, upon which the broker chooses the best search engines and forwards this query to these engines. Each search engine retrieves its most relevant results and sends these results to the broker. The broker merges these results into one result list and presents this to the user. Generally, the broker only controls the way the search engines are selected and the way their results are merged, it has no control over the internals of each search engine.

We propose a solution where specifically as many as possible Web hosts index their own content and thus become a (small) search engine. The key benefits of this approach include no or less crawling, as all or most servers index their own local content; more Web coverage; and more specialized search engines: these can be thought of as indexing structured data which would otherwise be hidden (e.g., inside the Deep Web), or which are specialized in finding relevant documents about a certain topic.

Before metasearch can be made operational, Callan [9] identified three major problems which must be solved first:

1. Resource description: describing the contents of each database;
2. Resource selection: given an information need and a set of resource descriptions, a decision must be made about which databases to search; and,
3. Result merging: integrating the ranked lists returned by each database into a single, coherent ranked list.

A resource description is often some kind of an excerpt of a database's index; it informs about the (estimated) number of different words in the index and how frequent these words appear in (some part of) the database.
A selection method is for example to treat each resource as one very large document and then to select that document with the highest query-term occurrence.
Simple methods for merging results are for instance to concatenate all results serially, or to combine the results in a Round Robin[1] (RR) fashion.

This research project is part of a bigger project on distributed Web search at the Twente University and focuses on the third problem of result merging.

1.2 Research Focus

This research aims at improving the efficiency and the performance of result merging methods. When a query is issued, we tackle the efficiency problems by restricting ourselves to use information only from the returned result pages, instead of downloading the documents either partially or completely. The simplest method using only such information is RR merging.

The problem of result merging can be viewed as that of re-ranking a set of ranked results. Ranking often involves combining multiple "sources of information" (e.g., the

[1] See Section 3.1

length of a document, the frequency of a word, the amount of similar terms in the query and the title, etcetera), which we call features. Not all features contribute equally to the result; they are often weighted.
Manual weighting becomes infeasible when dealing with many information sources. With enough computing resources and training data, techniques from the field of Machine Learning (ML) allow us to learn these weights automatically.
ML techniques are often used for classification and regression tasks. Nallapati [24], classifies documents as relevant or irrelevant. Others, [5, 13, 16, 18], classify pairs of documents (preference pairs), thereby indicating which document is more preferred (relevant) than the other. Finally, some researchers [26, 33] use regression to estimate global document ranks.
The Support Vector Machine (SVM) [39], a particular ML-technique, can be used for both classification and regression. It was shown that the SVM (trained on partial preference rankings) could be used to optimize the performance of a broker system and it even outperformed Google [18].
We will use the term preference-SVM to refer to the case where a classification model is trained on preference-pairs; and we will use the term regression-SVM to refer to the case where a regression model is trained on single training instances, not on pairs.

1.3 Research Questions

The main question this research will answer is:

> *Q1. Which of RR-merging, preference-SVM, and regression-SVM is recommended and why?*

For efficiency reasons, the solution to the result merging task is restricted at query-time: we are only allowed to download the result lists from a search engine, not the actual documents. However, since the broker first selects a number of search engines before merging their results, it must have some information about which engines are most capable of answering the query. We assume that this information is also available at query-time. Thinking in term of features, the next question is:

> *Q2. Using only information from the result lists and the broker's selection mechanism, what are suitable features to use for result merging, and what are their weights?*

It might happen that the broker selects a sub-optimal set of search engines, where, for example, the set contains too few engines that return relevant results. Ideally, the result merging strategies should produce a good ranking even with a sub-optimal set of search engines. This brings us to our next question:

> *Q3. How vulnerable are the merging strategies to external influences like the number of result lists to merge, or the quality of the result lists?*

Efficiency is gained by not downloading documents at query time. However, a more concrete indication of efficiency would be desired. The final question is:

> *Q4. How well do these result merging strategies perform in terms of the cpu-time / performance ratio?*

1.4 Thesis Outline

The following chapter introduces the ranking problem in Information Retrieval; it shows how features are generally used and aggregated in order to derive a ranking. Then, it introduces a specific approach for solving the ranking problem, called Learning to Rank, where it formally introduces the Support Vector Machine.
Chapter 3 gives an overview of the related work on Result Merging, and notes why certain approaches are not applicable for our experiments. Chapter 4 describes our research methodology; it describes our data, our result merging approach, the variables that were tested, and the evaluation procedure. Chapter 5 shows our results, these are discussed in Chapter 6. We conclude our work in Chapter 7 and discuss promising future work in Chapter 8.

Chapter 2

Ranking in Information Retrieval

Introduces necessary concepts to build upon and improve state-of-the-art ranking in IR.

2.1 Introduction

Information Retrieval concerns itself with the situation where a user, having some information need, performs queries on a collection of documents to find a set of relevant documents where the most relevant ones are ranked highest [15].

Traditionally, in (an) Information Retrieval (engine), documents are represented as a Bag of Words (BW) where the meaning of the document is simply seen as the collection of words it contains. This representation discards properties such as the structure of the text, word order, and much more. Furthermore, words in the document are often stemmed and this is optionally followed by removal of stop-words (e.g., non-content bearing words such as a, the, who), creating a bag of index terms. Likewise, the same pre-processing can also be applied to the user's query. Retrieval based on index terms fundamentally assumes that the semantics of the document and of the user information need can be expressed by sets of index terms [2].

Having representations of both query and document, the next step is to determine which documents are more likely to be more relevant to a given query. Today, almost all IR systems compute a single numeric score indicating how well a document matches the query. This is the result of aggregating values of features related to the document and/or the query terms. For example, term frequency, document frequency and document length are the main features used in many prominent (BW-based) IR models such as the Vector Space Model (VSM) [31], the Okapi BM25 probabilistic model [30] and language models [12, 20, 28]. To illustrate how features contribute to the computation of relevancy, an example will be discussed in the following section using the VSM.

2.2 Term Weighting; an Example

In the VSM, documents and queries are represented as feature vectors of terms that occur within the collection. The value of each element (feature) within the vector is called the (term) weight and is generally closely related to the term's frequency (TF) within the document.

Let us start with a simplified example. Imagine a salad recipe (i.e., a document) containing the four terms cabbage, tomato, fried, and bacon with term frequencies 3, 4, 2, and 5 respectively. Let us assume that our whole collection of documents contains only these four terms and that we put these features in the above order. This (i-th) document d would then be represented as:

$$\vec{d}_i = (3,4,2,5)$$

Queries can also be represented in the same way. For instance, the query q for fried bacon would be represented as:

$$\vec{q} = (0,0,1,1)$$

In the VSM, it is instructive to view both the documents and the queries as points (vectors) in a multidimensional space; the feature-values (or term weights) are the coordinates, and each feature is a different dimension. The intuition is that documents residing near the query can be seen as more relevant than documents that are farther away. Notice how the VSM does not define relevancy as a binary "yes" or "no". Instead, it specifies a distance between two objects where a shorter distance is assumed to imply higher similarity (and in this case, higher relevancy). The standard way of measuring the distance between these vectors is by taking the cosine of the angle between the object's vectors (Equation 2.1). As a result, similar objects will have a cosine similarity (sim for short) of one, while orthogonal objects (having no terms in common) will have a cosine of zero.

$$sim\ (Q, D_i) = \frac{\sum_j W_{Q,j} W_{D_i,j}}{\sqrt{\sum_j W_{Q,j}^2} \sqrt{\sum_j W_{D_i,j}^2}} \tag{2.1}$$

Of course, some terms may be more helpful than others may when determining the relevancy of a document to a given query. Terms appearing in only a few documents are more useful (in discriminating those few documents from other documents) than terms occurring in many documents across the collection. Weighting terms with their Inverse Document Frequency (IDF) is one way of indicating their discriminative power. The IDF of term t_i is simply the ratio N / n_i, where N is the total number of documents in the collection and n_i is the number of documents in which term t_i occurs. This form of combining TF with IDF is called TF.IDF weighting.

2.3 Distributed Collection Statistics

Collection statistics such as TF and IDF play a non-trivial role in discriminating relevant from irrelevant documents. TF.IDF somewhat samples the (importance of the) content of a web page. Other statistics such as PageRank [25], measure the quality and popularity of a web page. Note that for these statistics to achieve good performance, the collection information should be as complete as possible.

Whereas centralized IR has the luxury of gathering all crawled collection statistics at a central location, distributed IR simply cannot. If a search broker would gather all collection statistics of all remote search engines, besides almost becoming centralized IR, it would also involve huge amounts of bandwidth, definitely even more than centralized IR.

To minimize the network bandwidth consumption, the broker's problem of obtaining representative collection statistics should be solved for instance by using *highly discriminative keys* (HDK) [27, 36], or by estimating the remote collection statistics via query-based sampling (QBS) [10]. These methods are primarily used to rank the remote

search engines in their probability of returning most relevant results. QBS is also useful for result merging as can be seen in Chapter 3.

2.4 Learning to Rank

IR models with few parameters, such as the Okapi BM25, allowed researchers to hand-tune the model. However, fine-tuning features by hand becomes impossible when using many more features. Fortunately, with the recent availability of large standardized test corpora[1] and cheap computing resources, fine-tuning the features automatically has become possible.
Given a set of examples, the idea of Learning to Rank is to find those characteristics that can distinguish the good ones from the bad ones. Based on those characteristics, we should subsequently be able to predict whether new examples are good or bad.

Learning to Rank (LETOR) is a new and popular topic, both in Machine Learning (ML) and in Information Retrieval. In LETOR, the ranking problem is often formulated as a classification problem. We distinguish two subcategories: one (point-wise), a single document is classified as either relevant or not relevant; two (pair-wise), a pair of documents is classified, indicating which of the two documents is more preferred.

Close attention will be paid to the Support Vector machine (SVM) [39], which enjoys much popularity and is often reported with successful results. SVMs are a family of linear discriminant functions and are used for both classification and regression. A fact about SVMs is that they always find a global solution in contrast to neural networks, where many local minima usually exist [8].
Using point-wise classification, Nallapati [24] showed that SVMs are on par with state-of-the-art language models. A number of studies experimented with pair-wise classification, for instance [5, 13, 16, 18].

An SVM model is trained on some data sample $S = \{(\boldsymbol{x}^t, r^t) \mid \boldsymbol{x}^t \in \mathbb{R}^n, r^t \in Y\}$, where $\boldsymbol{x}^t$ is an n-dimensional feature vector[2] representing instance t, and r^t is the assigned label. We distinguish between classification if Y is a finite unordered set (nominal scale), and between regression if Y is a metric space, for example, the set of real numbers. The following subsections are mainly intended to give you an idea of what a support vector machine is and what it does.

2.4.1 Discriminant Functions

The simplest classifier is a linear classifier. A linear binary classification can formerly be written as $\text{sign}(f\colon X \subseteq \mathbb{R}^n \to \mathbb{R})$, with X a set of n-dimensional input data. So, each element $\boldsymbol{x} \in X$ receives a positive label if $f(\boldsymbol{x}) \geq 0$, and a negative label otherwise. In linear classification, $f(\boldsymbol{x})$ should be linear. For instance, it could be written as the dot product of the weight vector and the input vector, plus a constant:

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$$

[1] A collection of many documents and queries; for each query a number documents are judged by a group of people as being relevant or not relevant.
[2] See Section 4.2.

We must learn the right values of the parameters ($\boldsymbol{w}$, b), since these control our decision rule. In Figure 2.1, the thick line is a separating hyperplane as it separates the two classes x and o. It can also be seen that there are many possible separating hyperplanes. However, we want the hyperplane with the best generalizability, so that when given unseen data, it will most often produce the right classification.

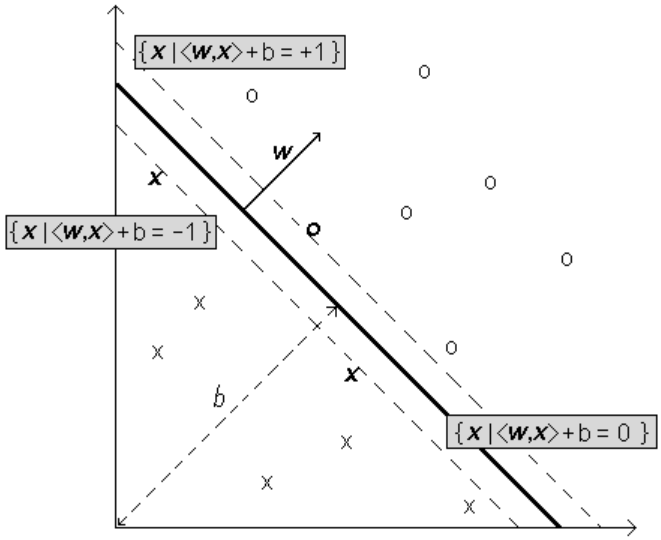

Figure 2.1: A two-dimensional space containing two linearly separable classes

A hyperplane could be chosen with the biggest distance to its surrounding data points. The distance from the closest point to the hyperplane is called the margin and so this hyperplane is called the maximal margin hyperplane.

2.4.2 Linear Support Vector Machines

Any hyperplane can be written as the set of points X satisfying $\langle \boldsymbol{w}, \boldsymbol{x}^t \rangle + b = 0$. Here, $\boldsymbol{w}$ is the hyperplane's normal vector, that is, it is perpendicular to the hyperplane. We want to choose ($\boldsymbol{w}$, b) such as to maximize the margin $p_{(w,b)}$ between the farthest possible parallel hyperplanes that still separate the data (see the dashed lines parallel to the hyperplane in Figure 2.1).

The canonical hyperplane is obtained by scaling ($\boldsymbol{w}$, b) so that for the nearest point $|\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b| = 1$ holds. Note that, if the data is linearly separable, the distance between the two parallel hyperplanes is 2. The margin is given by:

$$
\begin{aligned}
p_{(w,b)} &= \frac{1}{2}\left(\left\langle \frac{\boldsymbol{w}}{\| \boldsymbol{w} \|_2}, \boldsymbol{x}^+ \right\rangle - \left\langle \frac{\boldsymbol{w}}{\| \boldsymbol{w} \|_2}, \boldsymbol{x}^- \right\rangle \right) \\
&= \frac{1}{2 \| \boldsymbol{w} \|_2} \left(\left\langle \boldsymbol{w}, \boldsymbol{x}^+ \right\rangle - \left\langle \boldsymbol{w}, \boldsymbol{x}^- \right\rangle \right) \\
&= \frac{1}{\| \boldsymbol{w} \|_2}
\end{aligned}
$$

We can maximize the margin by minimizing the Euclidean norm $\| \boldsymbol{w} \|_2$. One of the reasons to prefer a maximal margin is the underlying assumption that both training and test data are drawn from the same distribution. Therefore, we could reasonably assume that a test point would be near a training example. If every test point is a maximum distance $r \geq 0$ away from a training point, then all test points will be classified correctly if we have a margin $p_{(\boldsymbol{w},b)} > r$.

When given a linearly separable data set $S = \{(\boldsymbol{x}^1, r^1) \ldots (\boldsymbol{x}^k, r^k)\}$, the (canonical) hyperplane $(\boldsymbol{w}, b)$ that solves the *primal* optimization problem

$$\begin{aligned} \textit{minimize}_{\boldsymbol{w},b} \quad & \frac{1}{2} \| \boldsymbol{w} \|_2, \\ \textit{with} \quad & r^t \left(\langle \boldsymbol{w}, \boldsymbol{x}^t \rangle + b \right) \geq 1, \; t = 1 \ldots k. \end{aligned} \tag{2.2}$$

will be a maximal margin hyperplane with margin $p_{(\boldsymbol{w},b)} = \dfrac{1}{\| \boldsymbol{w} \|_2}$.

In practice, researchers will often work on the *dual* representation[1], a Lagrange formulation of the problem. The reasons for doing this are two-fold. First, the constraints in (2.2) will be replaced by constraints on the Lagrange multipliers themselves. Second, the optimization problem can be expressed as dot products of its input vectors, such as in Equation 2.3. This makes it possible to apply the kernel trick, allowing us to train non-linear SVMs.

The dual optimization problem is a maximization problem:

$$\begin{aligned} \textit{maximize} \quad & W(\boldsymbol{\alpha}) = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{k} r^i r^j \alpha^i \alpha^j \left\langle \boldsymbol{x}^i, \boldsymbol{x}^j \right\rangle, \\ \textit{with} \quad & \textstyle\sum_{i=1}^{k} r^i \alpha^i = 0, \\ & \alpha^i \geq 0, \; i = 1..k \end{aligned} \tag{2.3}$$

Note that there is a Lagrange multiplier α^i for each training point. In the solution, the points for which $\alpha^i > 0$ lie on (one of) the hyperplanes and are called support vectors, as they 'support' the hyperplane. All other training points have $\alpha^i = 0$.

Given a solution $\boldsymbol{\alpha}`$ to the optimization problem and given a new data point $\boldsymbol{x}$, the choice which class to assign to $\boldsymbol{x}$ is obtained by taking its dot product with all the support vectors (remember that the α^i for the non-support vectors is zero anyway):

$$f(\boldsymbol{x}) = \operatorname{sgn}\left(\sum_{i=1}^{sv's} \alpha`_i \gamma_i \left\langle \boldsymbol{x}_i, \boldsymbol{x} \right\rangle + b` \right)$$

where

[1] See for instance Wellens [43] (written in Dutch) for obtaining this dual.

$$b` = -\frac{\max_{ri=-1}(\langle \boldsymbol{w}`, \boldsymbol{x}_i \rangle) + \min_{ri=1}(\langle \boldsymbol{w}`, \boldsymbol{x}_i \rangle)}{2}$$

$$\boldsymbol{w}` = \sum_{i=1}^{k} r_i \alpha`_i \boldsymbol{x}_i$$

2.4.3 Non-Linear Separable Data

Up until now, we have been assuming (noise free) linearly separable data, in which case it makes sense to choose a maximal margin classifier. As can be seen in Figure 2.2, a maximal margin classifier is not always the best option if the data contains noise (in this case, the noise is in the form of an outlier).

With non-linear separable data, a classification such as in Figure 2.3 will never be found by a maximal margin classifier because the constraint in Equation 2.2, that all data points should be classified correctly, is too strict.

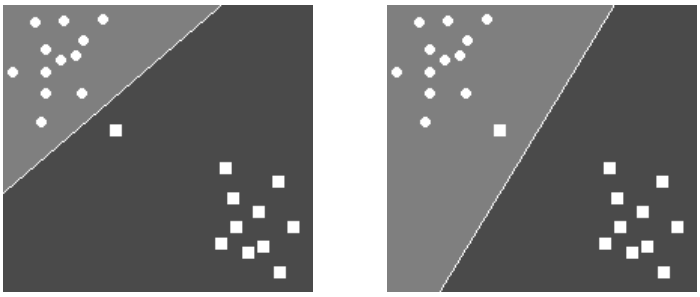

Figure 2.2: The left figure shows a maximal margin classification. The right figure has one misclassification, but is probably more desired. Based on similar figures in Wellens [43].

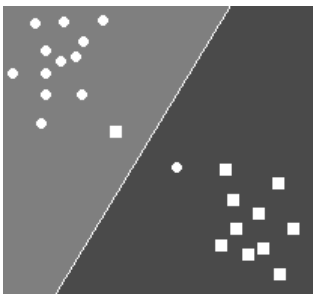

Figure 2.3: This classification cannot be made by our current maximal margin classifier because the data is not linearly separable. Based on similar figures in Wellens [43].

As can be seen in Figures 2.2 and 2.3, by allowing some errors, we can still make desired linear classifications. The constraints should somehow be relaxed. That is why Cortes and Vapnik [14] introduced positive slack variables:

$$\xi_t \geq 0\,,\ t = 1 \ldots k$$

which result in the following (weaker) constraints:

$$r^t\,(\langle \boldsymbol{w}, \boldsymbol{x}^t \rangle + b) \geq 1 - \xi_t,\ \ t = 1 \ldots k.$$

Each training instance gets a slack variable in such a way that the constraint is satisfied. Because of allowing classification errors, the margin would become infinite. Thus, the optimization problem should be modified to include a penalty for each error:

$$\begin{aligned} &minimize_{w,b} \quad \frac{1}{2} \| \boldsymbol{w} \|_2 + C \sum_{t=1}^{k} \xi_t \\ &with \quad r^t \left(\langle \boldsymbol{w}, \boldsymbol{x}^t \rangle + b \right) \geq 1 - \xi_t, \\ &\qquad \xi_t \geq 0, \, t = 1 \ldots k \end{aligned} \tag{2.4}$$

Here, C is a user-specified parameter and should be greater than 0. A bigger C means higher penalty on classification errors. Note that this algorithm will usually not result in a maximal margin classifier, which is why this is called a *soft margin classifier*.

The dual of Equation 2.4 is given by:

$$\begin{aligned} &maximize \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{k} r^i r^j \alpha^i \alpha^j \left\langle \boldsymbol{x}^i, \boldsymbol{x}^j \right\rangle, \\ &with \quad \sum_{i=1}^{k} r^i \alpha^i = 0, \\ &\qquad C \geq \alpha^i \geq 0, \, i = 1..k \end{aligned} \tag{2.5}$$

which is very similar to that of the maximal margin SVM in Equation 2.3.

2.4.4 Non-Linear Support Vector Machine

Perhaps the biggest showpiece of the SVM is that it can project the data points to some higher dimensional feature space and find a linear (soft margin) classification; usually corresponding to a non-linear classification in the input space. Kernels are used as the projection mechanism. Furthermore, kernels allow computational tractability when working in high or infinite-dimensional spaces.

Definition 1 (Kernel). A kernel is a function K, such that for all $\boldsymbol{x}, \boldsymbol{y} \in X \subseteq \mathbb{R}^n$

$$K(\boldsymbol{x}, \boldsymbol{y}) = \left\langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{y}) \right\rangle,$$

where Φ is a projection from input space X to feature space H.

Substituting the dot product for a kernel in Equation 2.5 yields the following optimization problem:

$$\begin{aligned} &maximize \quad W(\boldsymbol{\alpha}) = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{k} r^i r^j \alpha^i \alpha^j K(\boldsymbol{x}^i, \boldsymbol{x}^j), \\ &with \quad \sum_{i=1}^{k} r^i \alpha^i = 0, \\ &\qquad C \geq \alpha^i \geq 0, \, i = 1..k \end{aligned} \tag{2.6}$$

It is possible to encode extra a-priori knowledge in a kernel such that it can be used as a similarity measure. Even though kernels may differ, they should often be able (if they work well) to find roughly the same regularities in the given training data. This does not imply that it does not matter what kernel you use.

Next, the concept of the capacity of a learning machine, that is, its ability to learn any training set without error, will be explained by means of an example taken from Burges' tutorial on SVMs [8]. *A machine with too much capacity is like a botanist with a photographic memory who, when presented with a new tree, concludes that is not a tree because it has a different number of leaves from anything she has seen before; a machine with too little capacity is like the botanist's lazy brother, who declares that if it's green, it's a tree. Neither can generalize well.*

Intuitively, the use of a kernel will often be in accordance with increasing the capacity of the classifier.

When using SVMs for classification, two parameters must be specified: the trade-off parameter *C*, and the kernel. However, depending on the kernel, some additional parameters may have to be specified.

Examples of kernels:

1. Linear kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle$
2. Homogeneous polynomial kernels: $K(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle^d$
3. Inhomogeneous polynomial kernels: $K(\boldsymbol{x}, \boldsymbol{y}) = \left(\langle \boldsymbol{x}, \boldsymbol{y} \rangle + c \right)^d$
4. Gaussian radial basis function (RBF) kernel: $K(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma \| \boldsymbol{x} - \boldsymbol{y} \|^2 \right)$

The linear kernel requires no additional parameters. The homogeneous polynomial kernel has one additional parameter *d*. The inhomogeneous polynomial kernel has two additional parameters *c* and *d*. The Gaussian RBF kernel has one additional parameter gamma.

Chapter 3

Related Work on Result Merging

3.1 Introduction

The task of merging multiple result lists into a single ranked list is called result merging [33]. The result-lists are usually obtained by sending the same information need (often the same query) to N different (remote) search engines. There are no restrictions on the overlap-rates between the document collections of the different search engines, nor are there any restrictions on what ranking functions should be used.
The problem of result merging is not new and it has been a major open problem in distributed IR since around 1994 [33, 35].

Many terms have been used to refer to the problem of result merging, for example, data-, collection-, results-, information-fusion, and query-combination. Query combination is a more restricted definition of result merging. Particularly, it addresses the effect of merging result lists of different formulations of the same information problem to the same search engine [6]. This can be seen as a variant of query expansion where instead of producing a longer query, a set of N similar queries is produced. This set of N queries is sent to the same IR system and the resulting N result lists are then combined by, for example, using the weighted sum of the similarity scores.

Round Robin (RR) merging was briefly mentioned in Chapter 1. Because of its simplicity, it is often used as a baseline for merging experiments. RR merging is defined as follows: given n result lists $L_1, L_2 \ldots L_n$, take the first result r_1 from each list L_i as the first n results, then, take the second result r_2 from each list as the next n results, and so on. RR merging produces a list: $L_1r_1, L_2r_1 \ldots L_nr_1, L_1r_2, L_2r_2 \ldots L_nr_2, L_1r_3, L_2r_3 \ldots L_nr_3$, etcetera.
Often, the rank of the results is the only feature used when doing RR merging. The assumption is that all result lists have an equal distribution of relevant documents and that most relevant documents are ranked highest. However, any additional information about the relevant document distributions of the servers can be used to rank the servers. Having these two features, the rank of both the server and its results, RR will not blindly pick the next best result from a random server; it will pick the next best result from the next best server.

3.2 Merging Strategies

3.2.1 Normalizing Scores

In 1995 Callan et al. [11] tested four merging strategies: 1) interleaving (Round Robin), 2) raw scores, 3) normalized scores, and 4) weighted scores.

Search engines supplied numeric scores indicating how well the document matched the query, which enabled raw score merging. However, document scores from different search engines may often not be directly comparable. Normalizing statistics such as Inverse Document Frequency (IDF) could be a solution. The idea is that normalizing would achieve the same performance as when all different collections were combined in one global collection and then queried. Normalizing document scores entails significant communication and computational costs when collections are distributed across a wide-area network. Therefore, instead of normalizing the scores, one could weight them. Weights could be based on the document's score and/or the collection ranking information. Callan et al. showed that the performance of weighted score merging was as effective as normalized score merging, the other two approaches were significantly worse.

3.2.2 Clustering Techniques

Also around 1995, Voorhees et al. [42] developed two merging strategies that are independent of the IR-model of the different search engines. They defined the collection fusion problem as finding the values $\lambda_1 \ldots \lambda_c$, that maximize

$$\sum_{i=1}^{c} \lambda_i = N \text{ and } \sum_{t=1}^{c} F_Q^{I_t}(\lambda_t)$$

Here, N is the desired amount of documents to be retrieved, $F^I_Q(x)$ models the relevant document distribution of search engine I for query Q given x, the amount of documents to be retrieved. In practice, F^I_Q is not known and must be approximated. Their first strategy models relevant document distributions; the k most similar queries are used to learn a model of the relevant document distribution for each search engine I. These models are then used in a maximization procedure to learn the values λ_i. They use the VSM to compute similarities between queries. Their second strategy creates query clusters based on the amount of common documents retrieved, and assigns weights to these clusters. At query time, the most similar cluster to the query is selected from each search engine. Each cluster's weight, relative to the sum of all retrieved weights, determines the amount of documents to be retrieved from each search engine. Both strategies only determine the amount of documents to retrieve from each search engine. A total ordering on the result set is imposed either in a Round Robin fashion, or by chance: to select the document for rank r, a search engine is chosen by rolling a C-faced die that is biased by the number of documents still to be picked from each of the C search engines. The next document from that search engine is placed at rank r and removed from further consideration.

They show that these fusion techniques can approximate the performance of a single collection run at the ranks that will be of interest to the user.

3.2.3 Combining Evidence

In 2001, Rasolofo et al. [29] experimented with a current news metasearcher using low cost merging methods. They noted important differences between their news meta-searcher and a metasearcher of conventional search engines, one of which is:

> *In general, the titles of documents returned by current news services are more accurate and reliable than those typically available with other Web documents. Thus, titles may be a beneficial source of evidence in ranking articles;*

Their main merging approach was based on document scores, called raw-score merging. However, it was not practical since the document scores were seldom reported and they would not be comparable anyway due to differences in indexing and retrieval strategies used by the servers.
Therefore, they employed a generic scoring function that returns comparable scores based on various document fields (such as, title, summary, or date). For each document i belonging to collection j for the query Q, they compute a weight, denoted w_{ij} as follows:

$$w_{ij} = \frac{\mathrm{NQW}_i}{\sqrt{L_\mathrm{q}^2 + \mathrm{LF}_i^2}}$$

here, NQW_i is the number of query words appearing in the processed field of the document i, L_q is the length (number of words) of the query, and LF_i is the length of the processed field of document i.

They define several merging alternatives:

- their first alternative, which is also their baseline, is to apply RR merging based only on the ranks defined by the servers, denoted simply as RR;
- their second alternative is to first compute a score for each document using the XX field with their generic scoring function, and then adopt the raw-score merging approach. They denoted this alternative as SM-XX;
- their last alternative is to re-rank the results of each server using their generic scoring function, and then use the RR merging. This is denoted as RR-XX.

The scores are based on a combination of easily extractable information from result lists like: rank, title, summary, date. Additionally, they included estimated server usefulness and estimated collection statistics. Their best merging scheme (a raw-score merge based on a combination of estimated server usefulness, title and summary score) worked almost as well as merging based on downloading and rescoring the actual news articles.

3.2.4 Regression Models

Query-based sampling (QBS) can be used for building both descriptions of remote search engines (resources) and for building a centralized sample index. In 2002, Si and Callan [33] build resource descriptions with QBS and use CORI, a resource ranking algorithm, to select the 5 or 10 search engines with highest belief. The search engines return ranked lists along with document scores. After querying these selected search engines, they merge the result lists by learning a linear regression model to map the returned search-engine-specific document scores to the centralized sample index's document scores. They train one regression model for all search engines when the search engines are of the same type. When the types differ, they train separate regression models for each search engine. Si and Callan continued their experiments in 2003, and a more elaborated version can be found in [34]. CORI also merges results based on a linear combination of the document score and the search engine score. Si and Callan show that their regression model for merging results performs on par with the CORI result merging algorithm.
In 2007, Paltoglou et al. [26] go even further by regarding the sampled collections, obtained by QBS, not merely as descriptions, but as representatives for the remote search engines. They do not require the remote search engines to supply document scores. Instead, they execute the query both locally and at the remote search engine. The remote search engine returns a ranked list while the local (sampled) collection returns a ranked list along with the corresponding document scores. Applying regression analysis on the common documents on both lists, they can assign a score to each entry in the remote

result list. Once all remote result lists have been complemented with estimated document scores, they proceed just as Si and Callan [33] to map search-engine-specific document scores onto the centralized sample index's document scores. Paltoglou et al. show that their algorithm outperforms Si and Callan's regression method.

3.2.5 Experts and Voting

The merging strategies discussed so far are capable of merging results pages of search engines with varying degrees of collection overlap: they can be used even if there is no collection overlap.

Other merging strategies were developed specifically for cases with 100% collection overlap, that is, for search engines indexing exactly the same document collection. The intuition is that when every search engine is viewed as an expert, combining their different opinions would yield better results. Shokouhi [32] showed that combining these expert opinions often perform significantly better than the single best performing (expert) search engine.

Voting mechanisms are also used as a means to merge result lists. However, in order for voting mechanisms to work, the search engines should have some degree of collection overlap. One popular voting mechanism is the Borda Count [1]. The Borda Count assigns points to results as follows: for each search engine, the top ranked result is given c points; the second ranked result is given c-1 points, and so on. If there are some results left unranked by the search engine, the remaining points are divided evenly among the unranked results. The combined results are ranked in descending order of total points.

3.2.6 Download and Rank

Inquirus [21] follows an (extremely) impractical approach to result merging as it downloads the documents returned by the remote search engines and then re-ranks those documents; all of this happens at query-time.

Completely downloading the documents allows for more advanced operations such as better duplicate detection, better ranking, filtering of false results, etcetera. The negative aspects of this approach are higher bandwidth usage and longer delays in obtaining result pages.

3.2.7 Learning to Rank

As noted in Chapter 2, learning to rank is a popular topic in IR. Many researchers applied Machine Learning (ML) techniques to the problem of ranking, but, to the author's knowledge, only Joachims [18] applied SVMs to the problem of result merging. Joachims argues that clickthrough data is a rich source of "relevance-judgments" and that it can easily be obtained at practically no costs. The judgment is not a hard classification but a partial pairwise preference-judgment, indicating that one document is preferred over the other. Joachims first describes a modified SVM learning algorithm, SVM^{light} [17], which allows this preference data to be used as training data. He reported that his algorithm improved the performance of a broker system and outperformed Google.

3.3 Uncooperative Environments

With the enormous monetary incentives involved in today's search market, (remote) search engines that deliberately mislead the broker by providing falsified data should be

taken into account. When a search engine provides false data to the broker, the distributed IR system is said to operate in an uncooperative environment. In cooperative environments, all search engines provide faithful data to the search broker, and there can also be some form of centralized coordination of the search engines. For instance, the amount of collection overlap between the search engine, and their ranking algorithm(s) could be specified beforehand.

We maintain that the threat of uncooperative environments is always present. Suppose that the search engines *want* to be found, that is, they want to be used by the broker. One may argue that search engines, in an ideal case when a broker is able to detect and penalize unfaithful search engines, will *never* tamper with their collection statistics. However, in a not-so-ideal case, there is (much) incentive to provide false data.
Also note that, even in the ideal case, the case that search engines *never* tamper with their statistics holds only if there is one search broker; with competing brokers, incentives will again arise to hinder the competing search broker.

As for now, dealing with uncooperative environments remains an open problem. This research assumes operation in an uncooperative environment. For this reason, amongst others, we have restricted our selves to use only information from the result pages from the search engines, thus ignoring any meta-data that a search engine might provide.

3.4 Summary

Recall our research focus from Chapter 1: improving the efficiency and performance of result merging methods. For sake of query-time efficiency, we restricted ourselves to use information only from the result pages, more specifically, information that is generally accessible to the user.
In addition, we argued in Section 3.3 that any information other than what a user normally sees, such as raw-document-scores, should be regarded as biased and potentially misleading. Furthermore, Callan et al. (see Section 3.2.1) discourage the use of raw-document-scores for result merging unless they are weighted. However, weighting can entail significant amounts of computational and communication costs.
Finally, downloading documents for whatever reason is an additional cost that we want to avoid as much as possible.

Regarding our research, many of the strategies in Section 3.2 are not applicable. We do not download any document; we cannot normalize document scores, as we do not have these; and, we cannot use voting mechanisms, as there is no collection overlap in our scenario.
We restricted ourselves to using information only from result pages and from the broker's selection mechanism. Results of search engines have a rank, a title, a snippet, and a URL; therefore, applicable strategies for result merging are the RR merge and ML methods such as (pairwise) classification and regression.
Although it has been shown in numerous studies that RR merging was not the best solution, it serves as a baseline indicating the minimum search performance that our SVM-strategies should be able to achieve.

Chapter 4

Research Methodology

This chapter explains how the merging methods were implemented, tested, and evaluated.

4.1 Dataset

Our experiments were conducted using the TREC WT10g corpus, which was created in 2000 [4]. The WT10g is a carefully engineered selection of the larger 100-GB VLC2 collection, which is a truncated Internet Archive Web-crawl from February 1997. The WT10g collection was devised to be broadly representative of Web data in general; to contain many inter-server links; to contain all available pages from a set of servers; to contain an interesting set of metadata; and to contain few binary, duplicate or non-English documents.
A test collection also requires a set of queries and relevance judgments. The WT10g collection has human-made relevance judgments for 100 ad hoc relevance topics (used for querying). These judgments were based on pooling[1] and were classified as irrelevant, relevant or highly relevant. The topics were reverse engineered by NIST from the log files of web search engines; they include the original Web query in the title field. Topics 451-500 include a number of misspelled words whereas topics 501-550 do not.

Although the WT10g corpus is a multi-purpose test collection for Web retrieval experiments, it was not necessarily created for Distributed Information Retrieval. The result merging experiments require result pages from different search engines which index different documents. The WT10g collection does not include any form of result pages so these had to be created first.
The MonetDB/XQUERY database system has the capability to create result pages with each result having a rank, title, snippet and URL. A number of steps had to be taken in order to create an environment where a number of search engines index disjoint (but not necessarily covering) subsets of the whole WT10g document collection. Each of these steps will be explained in the next subsection.

4.1.1 Creating Subcollections and their Result Pages

The MonetDB/XQUERY retrieval platform requires its data to be valid XML. Thus, the first step was to convert all WT10g data into valid XML. Therefore, a script was used that: 1) discarded the HTML comments, scripts, and all but the title and anchor HTML-tags; 2) truncated URLs ending in "/index...." at the index-portion; 3) glued consecutive sentences shorter than 20 characters together and split sentences longer than 160

[1] A pool of documents is created from the top N documents submitted by TREC participants. (All participants create different search engines, and they often have different results.) Only documents in this pool are judged by human assessors.

characters; finally, 4) in the case that a document did not have a title, a title was created from the first sentence of the document.

The documents (web pages) in the original WT10g corpus were randomly distributed over several file chunks. It is assumed that the pages of a website are highly related to each other and that they most often reside on the same web server. This led to the second step of re-grouping the documents by their IP-address, which resulted in XML-documents containing all web pages of a single server. We will refer to these newly created documents as ip-split documents.

The third step is to create subcollections from these ip-split documents. A simple set of rules was used to create these subcollections. First, the ip-split documents were sorted by their file size. Then, each ip-split document *ipdoc* was added to a subcollection if the combined size would not exceed a specified size of *X* MB. Otherwise, a new subcollection was made containing *ipdoc*. Note that an ip-split document bigger than *X* MB was not split. In pseudo-code:

Table 4.1: pseudo-code for creating collection splits

```
INITIALIZE docs to the sorted ip-split documents
INITIALIZE subcollections to an empty collection
INITIALIZE sub to an empty collection

FOREACH doc in docs
   IF (sub is empty OR size(sub+doc) ≤ X) THEN
      ADD doc to sub
   ELSE
      ADD sub to subcollections
      EMPTY sub
      ADD doc to sub
   ENDIF
ENDFOR

ADD sub to subcollections
```

A number of subcollections were made based on a 100MB and 500MB split. Splitting the WT10g collections in chunks of roughly 100MB resulted in 79 subcollections. Similarly, splitting in chunks of 500MB resulted in 15 subcollections.

The fourth and final step indexes each subcollection by a (separate) search engine, and queries the search engine to get the result lists (with a maximum of 50 results) needed for the result merging experiments.
The MonetDB/XQUERY database system was used to make separate indices for each subcollection. In addition, each query was issued twice; first using the OKAPI BM25 IR-model, and then using the Normalized Log-Likelihood Ratio (NLLR) IR-model.

Figure 4.1 illustrates the process of making these result pages.

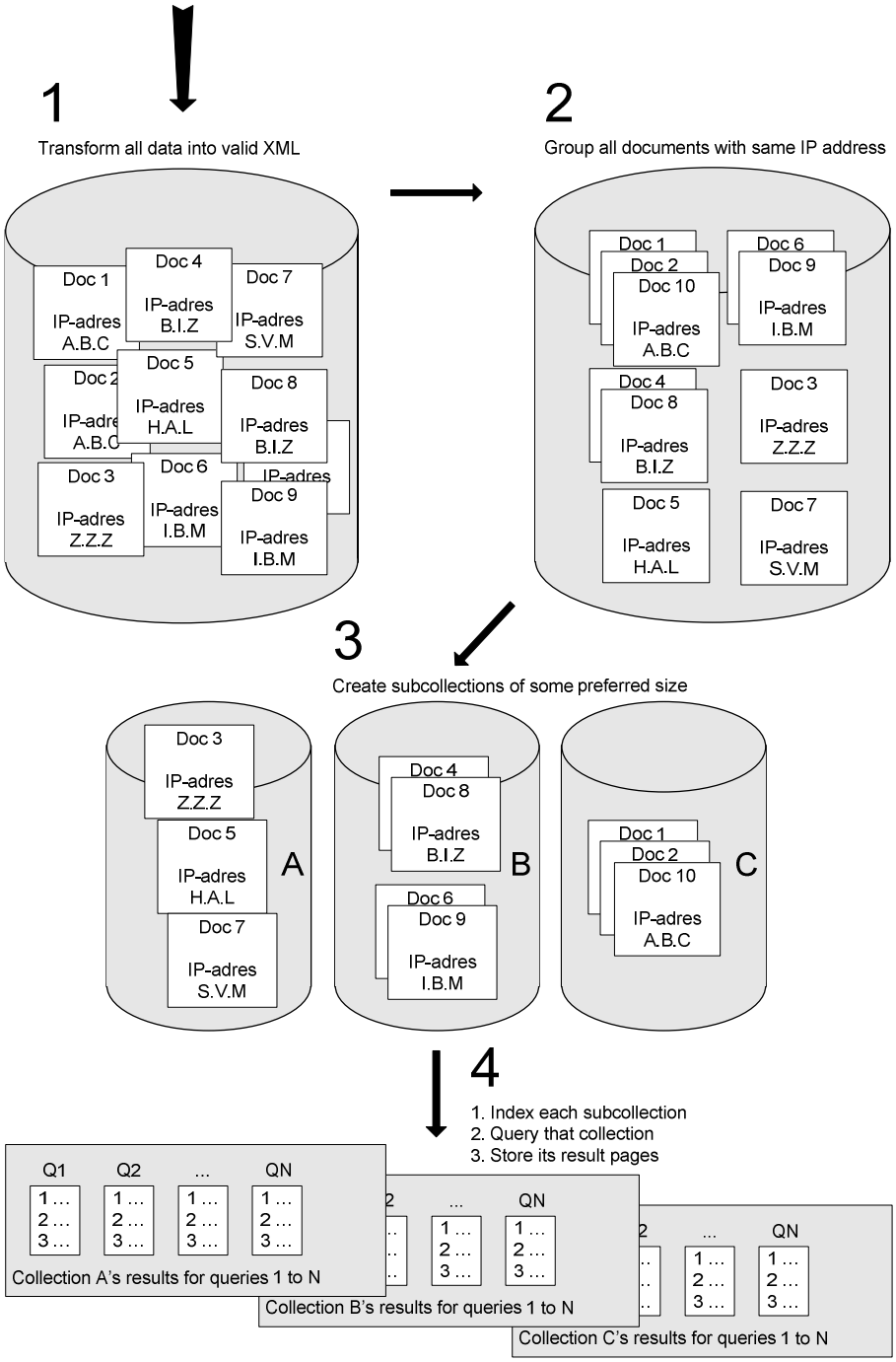

Figure 4.1: simulating the creation of result pages in a distributed environment

Summarizing, after performing all these steps, the following data sets were created:

- 79 subcollections (of +/- 100MB), each subcollection containing:
  - 100 different result pages created with the OKAPI IR-model
  - 100 different result pages created with the NLLR IR-model
- 15 subcollections (of +/- 500MB), each subcollection containing:
  - 100 different result pages created with the OKAPI IR-model
  - 100 different result pages created with the NLLR IR-model

4.1.2 Selection Policies

When doing result merging, it is not the intention to merge all the results from all search engines; otherwise, one could as well have used a centralized index[1]. This means that a selection must be made about which search engines to use for merging. The selection can affect the results of the merging experiments; it is very likely that a random selection and one based on the best performing search engines for a particular query would yield significantly different merging results.

This raises the question for a suitable measure of a search engine's performance. Let us pay closer attention to the well-known Average Precision (AP) measure [41]. For a given query, the equation for the AP of a ranked result list is:

$$\text{AP} = \frac{\sum_{rank=1}^{N} P(rank) \bullet R(rank)}{number\ of\ relevant\ documents}$$

where N is the number of retrieved results, $P(x)$ gives the precision[2] at rank x, and the binary function $R(x)$ is 1 if the result at rank x is relevant or 0 otherwise. The AP can take as output values any value between zero and one. The Mean Average Precision (MAP) is obtained by averaging over multiple queries' AP.

The WT10g relevance judgments contain a list of relevant documents per query; call these the global relevant documents. The WT10g collection is divided into several subcollections and we will refer to the number of relevant documents in such a collection as the local relevant documents. Figure 4.2 gives a simple example.

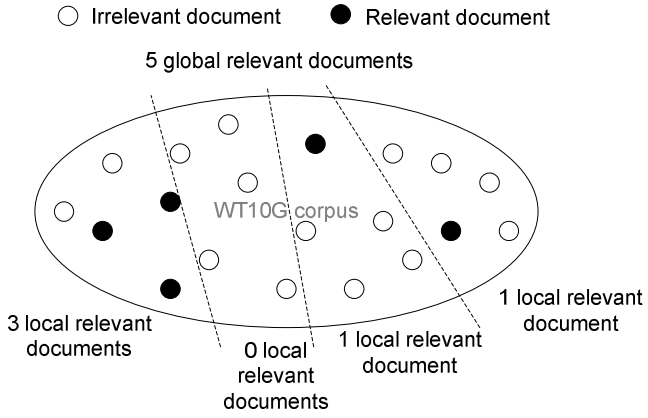

Figure 4.2: global versus local relevant documents.

Thus, for a given subcollection, the AP measure can be calculated by using either the global or the local relevant documents. The terms LAP, GAP, LMAP, and GMAP will be used to refer to local or global AP or MAP.

---

[1] Under the following assumptions: first, a centralized IR system should perform at least as good as a Distributed IR (DIR) system, since it has complete knowledge of the collection. Second, many search engines in a DIR environment contribute no relevant results.

[2] The precision is defined as the number of relevant documents retrieved (at rank x), divided by the total amount of documents retrieved at rank x (i.e., x).

Other ways of selecting search engines are random selection and selection based on the engine's *merit*, the number of relevant documents it contains. Keep in mind that these are not performance measures.

4.2 SVM Models for Result Merging

An SVM model is affected by its training data and by its parameters. Sections 4.2.1 up to 4.2.4 explain how the results were converted to training data. Section 4.2.5 explains which parts of this data were actually used for training the SVM models, and how the parameters were varied.

4.2.1 Features

SVM models are trained with labeled feature-vectors; these were extracted from the result lists for each query. Table 4.2 lists the thirty features used in our experiments; the range of each feature-value is given in the parenthesis at the end of the line.

Table 4.2: grouped feature list

1	1. Rank: ratio (1000 – rank) / 1000 (range [0,1]) 2. Rank: local rank (range [1…])
2	3. Cosine similarity query – title (range [0,1]) 4. Cosine similarity query – snippet (range [0,1]) 5. Cosine similarity query – URL (range [0,1])
3	6. Title: number of words (range [1…]) 7. Title: average word length in chars (range [1…])
4	8. Snippet: number of words (range [0…]) 9. Snippet: average word length in chars (range [1…])
5	10. URL: length in chars of FQDN (range [4…]) 11. URL: FQDN frequency in current list (range [0,1]) 12. URL: path-depth (e.g. http://a.b.c./depth=1/…) (range [0…]) 13. URL: average path length in chars (range [1…])
6	14. Query: number of words (range [1…]) 15. Query: average word length in chars (range [1…])
7	16. URL: contains tilde '~' (binary {0,1}) 17. URL: contains text 'home' (binary {0,1}) 18. Title: contains text 'home' (binary {0,1}) 19. LCS query – title (range [0,1]) 20. LCS query – snippet (range [0,1]) 21. LCS query – URL (range [0,1]) 22. LWO query – title (range [0,1]) 23. LWO query – snippet (range [0,1]) 24. LWO query – URL (range [0,1]) 25. LWO title – snippet (range [0,1]) 26. LWO title – URL (range [0,1]) 27. LWO URL – snippet (range [0,1])
8	28. Server usefulness: LAP (range [0,1]) 29. Server usefulness: GAP (range [0,1]) 30. Server usefulness: Merit (range [0…])

The abbreviations LCS and LWO respectively denote Longest Common Substring and Longest Word Order. These features were introduced in order to compensate for the information-loss caused by the IR-models, NLLR and OKAPI, which are used by the search engines. These IR-models build upon the Bag of Words concept, which discards many properties of text.

LCS(A,B) detects the biggest unaltered proportion of A that also appears exactly the same way in B. LWO(A,B) is almost similar to LCS, but it allows for noise. For example, let A denote the text "*using ranking SVM in IR*" and let B denote "*using Machine Learning techniques for ranking in IR*". The LCS similarity between A and B is low (0.4), while we would expect otherwise. The LWO similarity does capture this apparent resemblance of A and B, yielding an LWO of 0.8. The pseudo code for LCS and LWO can be found in the Appendix A and B. The last three features are intended as a simple measure of the coherence of a result; do the title, snippet, and URL somewhat resemble each other?

Having extracted the feature vectors, we can now label them. A label is simply a digit and is placed at the start of each feature-vector. First, the concept of preference pair constraints will be explained, and then the appropriate labelings will be discussed for both the preference-SVM and regression-SVM.

4.2.2 Preference Pair Constraints

Clickthrough data is a cheap source of relative relevance judgments and is used to create preference pair constraints. These constraints are used for training an SVM model (which should optimize the rankings of a search engine). Consider the result list in Table 4.3.

Table 4.3: example result list

```
1. Cattery Lopend Vuur
   http://www.xs4all.nl/~bengaal

2. Supreme Show "Club Row" Stands
   http://www.cityscape.co.uk/users/ja49/supclub.html

3. Tejas Bengal Cats
   http://www.io.com/~tejas

4. Tejas Cattery
   http://www.io.com/~tejas/whatis.htm

5. Nerd World : CATS
   http://search.nerdworld.com/nw460.html
```

Suppose we are using clickthrough data and that a user actually clicked on results 1, 3, and 5. Joachims [18] argues that it is not possible to infer that results 1, 3, and 5 are relevant on an *absolute* scale; however, it is plausible to infer that result 3 is more relevant than result 2 with a probability higher than random. *Assuming that the user scanned the ranking from top to bottom, he must have observed result 2 before clicking on 3, making a decision not to click on it.* In other words, the search engine should have ranked result 3 higher than 2, and result 5 higher than 2 and 4. Denoting the ranking preferred by the user with r*, we get the following (partial and potentially noisy) preference constraints:

$$\text{result}_3 <_{r^*} \text{result}_2 \qquad \text{result}_5 <_{r^*} \text{result}_2 \qquad \text{result}_5 <_{r^*} \text{result}_4 \tag{4.1}$$

In our experiments, TREC relevance judgments are used as our source of preference constraints. Note that clickthrough data, as a source of relative relevance judgments, and the TREC relevance judgments differ in several ways. First, clickthrough data is relative

and is based on superficial information supplied by the search engine (for example, ranks, titles, snippets, and URLs). Even if a result is judged more relevant (because a user clicked on it, but not on a higher ranked result), it does not mean than the result is actually relevant. TREC judgments are *absolute*; a team of people have actually read the entire document and rated that document as being irrelevant, relevant, or highly relevant. This brings us to the second difference: clickthrough data is "binary" (a result is either clicked on, or not) whereas TREC judgments are ternary.
Using TREC relevance judgments, consider again the result list in Table 4.3, where results 1, 3, and 5 are relevant, and where result 4 is highly relevant. The search engine *should* have ranked those results as follows: 4, 1, 3, 5, 2. The constraints are:

$$\begin{array}{lll} result_3 <_{r^*} result_2 & result_5 <_{r^*} result_2 & result_4 <_{r^*} result_1 \\ & & result_4 <_{r^*} result_2 \\ & & result_4 <_{r^*} result_3 \end{array} \qquad (4.2)$$

A final note on the use of preference pairs: Joachims states that for each clicked result, he adds fifty additional constraints indicating that it should be ranked higher than a random other result in the result set. The rationale is that those constraints should hold for the optimal ranking in most cases and they should both stabilize the learning result and keep the learned ranking function close to the original ranking function.
His result set consisted of 500 results. Since our result set consists of only 250 results, we used thirty instead of fifty additional constraints.

4.2.3 Preference-SVM Labeling

SVM*light* automatically creates the constraints based on the labels of the training data. An additional feature 'qid' is used to restrict the generation of the constraints; SVM*light* considers any two instances for a preference pair constraint only if they have equal 'qid' values. Such preference pairs are interpreted as follows: the instance with a higher label value should be ranked higher than the other one. For example: we could extract the feature-vectors from the results of Table 4.3, and label them as follows:

```
0 qid:1 (feature-vector for result 2)
1 qid:1 (feature-vector for result 3)
0 qid:2 (feature-vector for result 1)
0 qid:2 (feature-vector for result 2)
0 qid:2 (feature-vector for result 3)
1 qid:2 (feature-vector for result 4)
0 qid:3 (feature-vector for result 2)
1 qid:3 (feature-vector for result 5)
```

In the example, we see four instances having a 'qid' of two. If we create all possible pairs from these four instances, then those pairs with unequal labels are the preference constraints. Verify for yourself that the appropriate constraints (4.2) can be created based on the labels in the example above.

The pseudo code for finding the preference pair constraints and label them accordingly can be found in Appendix C.

4.2.4 Regression-SVM Labeling

With regression-SVM, we aim to train a model that, given a feature-vector, will output a global (artificial) rank. We want this rank to reflect the knowledge obtained from both the TREC relevance judgments and from the rankings of the search engines. However, the TREC judgments should have a higher impact on the learned ranking function. For instance, if a highly relevant result (according to the TREC judgment) was ranked lowest by some search engine, then we certainly want our new ranking function to rank that result somewhere near the top.

From irrelevant to highly relevant, we denote the TREC judgments as 1, 2, and 3. We created the labels of the training instances by multiplying the instance's relevance judgment by 1000, and then deducting its rank. This label, a global artificial rank, reflects the knowledge obtained from the TREC judgments and from the rankings of the search engines, while ensuring that the TREC judgments have a higher impact.

4.2.5 Model Settings

Preliminaries: Data Preparation & Feature Selection

Our training data consists of a subset of the results that were created in Section 4.1. Several subsets, or data chunks, could be used for training, for example, we could use the results of one search engine, or of three, or all, as our training data.

We used the result pages from the 100MB or the 500MB collection splits, indexed with either the NLLR or the OKAPI IR-models. We trained on the results of 1, 4, 7, 10 or 15 search engines, and we used the following policies to select the search engines: LAP, GAP, random, and merit.

When training on data from more than one search engine, we first applied a RR merge. Unfortunately, as the amount of training samples increased, so did the time required to train the SVM models. Therefore, we decided to merge no more than 250 results in a RR-fashion. Due to the large amount of time needed for training SVM models, we did not experiment with all the possible combinations using the OKAPI IR-model.

We divided our thirty features into eight feature groups (see the feature list, Table 4.2). Since we could not be certain whether some feature actually introduced noise or not, we did a simple feature selection experiment. In this experiment, we repeatedly trained a model while only changing one of the feature groups. That is, for each data chunk, we trained both a regression- and a preference-SVM model using either: all feature groups, or, each combination of seven out of all eight groups. A linear kernel was used in these preliminary experiments.

The preliminary experiments give an indication of the optimal data settings for learning a good SVM model, that is, they roughly indicate which data chunks and feature groups are best for either regression or preference SVM learning. Part of these results is shown in Chapter 5, Section 5.1.

At first, we wanted to optimize the SVM parameters for the top three data settings, which are listed below:

- Regression: 100MB, OKAPI, 1 search engine, LAP, all features except group 3
- Regression: 100MB, NLLR, 10 search engines, GAP, all features except group 3
- Regression: 100MB, NLLR, 10 search engines, GAP, all features except group 7

However, since there is only data from the 100MB split and since only regression models seem to perform best, we decided to use the following additional data settings, the best 500MB data setting and the two best settings for the preference-SVM:

- Regression: 500MB, NLLR, 7 search engines, LAP, all features except group 7
- Preference: 100MB, NLLR, 4 search engines, LAP, all features except group 3
- Preference: 100MB, NLLR, 4 search engines, LAP, all features except group 7

SVM Parameter Optimization

One parameter, which always applies, is the complexity factor C. This determines the trade-off between minimizing model error on the training data and minimizing model complexity. Its purpose is to avoid overfitting and underfitting. Overfitting occurs when the model performs well on the training data but does poorly on held-aside test data. Underfitting occurs when the model performs poorly on both training and test data.
Another parameter is the kernel. Depending on the type of kernel used, additional kernel parameters must be specified[1]. Finally, in case SVM regression is used, an additional parameter epsilon must be specified; its purpose is to penalize large errors and neglect small ones.

For our experiments, we varied the *C* parameter from two to the power of -2.5, -2, -1.75, -1.5, -1, 3, and 6. We used both a linear kernel and a Radial Basis Function (RBF) kernel. The gamma parameter for the RBF kernel was varied from four to the power of -1, 0, 1, 2, and 3. Finally, in the case of training regression models, the epsilon parameter was varied from five to the power of 0.5, 1, 1.5, 2, and 2.5.

Summarizing, the preliminary tests concluded that four data chunks were suitable for training regression models: for each chunk, we supplied 210 different parameter-combinations to the SVM algorithm. This resulted in 210 regression models: 35 with a linear kernel and 165 models with an RBF kernel. For each of the other two data chunks, we created 42 preference models: seven with a linear kernel and thirty-five with an RBF kernel.

We do not expect all 924 (4 x 210 and 2 x 42) models to generalize well, and we will discuss the evaluation of the SVM models in Section 4.4.1.

4.3 External Influences

We set out to analyze the effects of other variables, which we assumed to have great influence on the merging performance of our strategies.

4.3.1 Number of Result Lists

Merging more result lists inherently enhances recall. How this affects precision is hard to say. For example, with RR merging, precision is very much affected by the *quality* of a result list: it really matters that the first few results in each list are relevant. In many lists however, the first result is not relevant (that is, there are many lists of low quality) and no result list consists of only relevant results. Assuming that each time the list with the highest *quality* is selected next, then, as more result lists are being merged, precision will most likely increase until some optimum is reached, and then start to decrease.

[1] Examples of kernels and their parameters were explained in Section 2.4.4.

By varying the amount of lists to be merged from 2, 3, 4, 5, 7, 10 and 15, we measured the precision-stability of each merging algorithm.

4.3.2 Subcollection Size

As the size of the subcollections increases, the number of the subcollections decreases. This has one major impact: the subcollection's collection statistics are directly affected, influencing the IR-model's performance and thus influencing the quality of the result pages.

A minor effect is that, as the size of the subcollections increases, the proportion of documents included in the search increases when the number of result lists to merge stays the same. For example, when merging 15 result lists, we are searching 100% of the documents if we use our 500MB collection split consisting of 15 subcollections.

4.3.3 Search Engine Selection Policy

Section 4.1.2 explained that Local/Global (Mean) Average Precision can be used as performance-based selection policies. In addition, two non-performance-based selection policies were explained: random selection and merit-based selection. We measured the effects of these six selection policies on the merging performance.

4.4 Evaluation

4.4.1 Evaluating SVM Models

We want to find out which model is the most *robust* model: that is, the one that produces the highest GMAP as often as possible regardless of external influences: the resource selection policy used, the amount of search engines merged, and the size of the search engine's document collection.

First, we split the result pages from each search engine for each query in two parts: the results of the odd queries for training and the results of the even queries for testing. However, not all results of all odd queries were used for training and not all results of all even queries were used for testing; instead, several data chunks were used. As discussed earlier in Section 4.2.5, six data chunks were selected: four chunks, each for training 210 regression models; and two chunks, each for training 42 preference models. The models were trained on the chunk's odd queries (the training set) and evaluated on the chunk's even queries (the test set).
The evaluation consisted of measuring the model's performance on the test set by means of the LMAP[1] measure. An N-fold cross validation would have been a more accurate estimation of the model's performance; however, it would have cost considerably more time.

The test set evaluation indicates which models generalize well, however, it is not sufficient to conclude which model is the most robust model. For example, suppose that we used a data chunk, created from 3 search engines from the 100MB split according to

[1] On a side note, for each training set, we trained either 210 or 42 models. Since the test set remains fixed, it does not matter if the LMAP or GMAP measure was used; the only difference would be that the GMAP would be lower than the LMAP by some constant value.

LAP selection, for training and testing two models A and B. According to the test set evaluation, the best model is A. It might be the case that A performs better when merging results from 2 and 3 search engines, whereas B performs better when merging results from 4, 5, 7, 10 and 15 search engines. Since B performs better in most cases, we define B as the most robust model.

In order to find the most robust model, we needed to see how the models would react to varying external influences. Recapitulating, the external influences were the number of search engines to merge: 2, 3, 4, 5, 7, 10, and 15; the selection policies: LMAP, GMAP, GAP, LAP, random, and merit; the collection split size: 100MB and 500MB; and finally, the IR-models: NLLR and OKAPI.
There are 148 different value-combinations of these external variables. To test each of the 924 models on all 148 combinations would be very time consuming. Therefore, we used the top ten models from each of our six training sets, based on their test set evaluation. We assumed that the most robust model would also be among the top ten models in the test set evaluation.

For each combination of external variables, we applied the sixty models and we ranked them based on their performance on that particular experiment, the best model at rank one, the second best model at rank two, etcetera. Once all 148 experiments were conducted, the mean rank of each model was calculated. The model with the best mean rank was considered the most robust model.

4.4.2 Evaluating Retrieval Performance

Statistical Significance Test

To test whether the SVM models were significantly (with $p<0.05$) better than the RR baseline merging method, we used a randomization approach [37] with 100,000 random permutations. Our test statistic was the GMAP of each method.

The randomization test is designed to determine if the observed difference in the MAP of two systems A and B is due to the systems or to chance. The null hypothesis H_0 states that both systems are identical. We can imagine some system N generating two Average Precision (AP) values, one labeled A, and the other labeled B. Given 50 topics, there are 2^{50} possible ways to label the APs; one of those labelings is exactly the observed labeling of our systems A and B.
Under H_0, any permutation of the labels is equally likely. If all 2^{50} permutations were created, and if we measured the difference in the MAP of A and B for each permutation, we could count the number of times the absolute difference in MAP between A and B was as great as or greater than their observed absolute difference. This number divided by 2^{50} yields the two-tailed p-value. If this value is less than 0.05, we reject H_0 and we are confident that the GMAP difference is due to the systems and not due to chance.
In practice, generating all possible permutations would take too long even for modern machines; therefore, we generate 100,000 random permutations and we assume that the two-tailed p-value obtained this way to be accurate enough.

Comparing with Centralized Baselines

A comparison with a centralized baseline is the ultimate test for a Distributed IR system, since the former has complete knowledge of the whole collection, allowing it to achieve the highest performance.
The MonetDB/XQUERY database system was used as our centralized baseline. The WT10g corpus was converted to valid XML, just as described in Section 4.1.1. Then, an index was made on all the data in order to retrieve the desired result pages. Each result page had a maximum of 500 results.
The GMAP measure is well suited for this comparison, since we are not comparing any merging performance (what does a centralized system merge?); instead, we are comparing the overall retrieval performance of both systems.

A Side Note: Comparing Merging-Performance

It is not clear what measure to use as a merging-performance measure. When comparing two different distributed IR systems, one often cannot conclude, based only on the GMAP, that one merging scheme merges better than the other does; however, one can conclude that the whole system, thus including the resource selection component, did a better job of choosing its resources and merging the results.
The following example illustrates the problem. Imagine three merging systems A, B, and C having perfect recall (retrieving all relevant documents in their resources) and producing a ranking where all relevant results are in the top even ranks: for example, all three systems have an LMAP of 0.5 regardless of how many relevant documents reside in their document collection. Now imagine system A having 10, B having 20, and C having 50 relevant documents. While their LMAP is the same, their GMAP is 0.0625, 0.125, and 0.3125 respectively.

The LMAP does indicate that the three systems, given their situation, were equally well in merging their results, whereas GMAP certainly failed this task. Even so, it is questionable whether the LMAP measure can be used as a merging-performance measure; who is to say that system A, given 50 relevant documents in its collection, would have produced a ranking "as good" as system C? Maybe it would have performed a little better, or maybe even worse.

The only way to compare merging performance between systems is if all else remains fixed: the test corpus, the resource selection component, the IR-models used for indexing that resource, etcetera. When this is the case, it does not matter anymore whether LMAP or GMAP is used to compare the merging-performance.
Although the problem of result merging has been acknowledged since 1994, no "official" merging-performance measurement yet exists. This means that, unless a lot of effort was put into duplicating the experiment-environment and only changing the merging algorithm, people were never really optimizing the merging algorithm itself; instead, the system as a whole was being optimized. Therefore, whenever progress was made, it could have been attributed to the resource selection component, or the merging component, or both.
It might be useful to supply both LMAP and GMAP figures for each merging experiment, thereby providing a better means for comparing the merging performance of different Distributed IR systems.

Chapter 5

Results

In this chapter, we report and analyze our results. First, we report part of the results of our preliminary tests. Then we show the performance of the centralized baselines. Then, we show the SVM training and merging times, and compare them with the RR merging time. Thereafter, we present the resulting features of the best SVM model and their corresponding weights. Finally, we present performance charts of SVM and RR merging, while we vary the external influences.

5.1 Preliminary Test Results

The preliminary tests were carried out to find out which data chunks and feature groups were suitable for further optimizing either the preference-SVM models or the regression-SVM models. For each data chunk, a preference-SVM and a regression-SVM model was trained using a linear kernel. All models were evaluated on how they merged the results of the top five search engines, selected with the GAP policy, the results of which are shown in Table 5.1.

Table 5.1: GMAP of the SVM model when merging the results of the top five GAP search engines

GMAP	SVM-type	Search engines	Selection policy	Collection size	IR-model	Feature group omitted
0,194	regression	1	LAP	100MB	OKAPI	3
0,191	regression	10	GAP	100MB	NLLR	3
0,191	regression	10	GAP	100MB	NLLR	7
0,191	regression	10	GAP	100MB	NLLR	None
0,190	regression	10	GAP	100MB	NLLR	4
0,190	regression	10	GAP	100MB	NLLR	6
…						
0,189	regression	7	LAP	500MB	NLLR	7
0,189	regression	10	GAP	500MB	NLLR	6
…						
0,151	preference	4	LAP	100MB	NLLR	3
0,142	preference	4	LAP	100MB	NLLR	7
0,142	preference	4	GAP	100MB	NLLR	4
0,140	preference	4	GAP	100MB	NLLR	3
…						

We used the top three settings for further parameter optimization. In addition, since these are all from the 100MB collection and are all regression models, we also used the best 500MB setting and the two best settings for preference-SVM models for further parameter optimization.

Simply said, parameter optimization means training many distinct SVM models on the same training data and finding out which one is the most robust model. We discussed our evaluation procedure in Section 4.4.1.
The most robust SVM model turned out to be a regression-SVM model, with a linear kernel, trained on the best data setting (i.e., the first row in Table 5.1): the odd query results from the search engine with the highest LAP (for that query), from the 100MB OKAPI split, with all but the title features.

5.2 Centralized Baseline Performance

Using the fifty even-numbered queries for evaluating our centralized baselines, where each result list has a maximum of 500 results, the MAP is given in Table 5.2.

Table 5.2: MAP for centralized baselines using even-numbered queries

	MAP@500	MAP@250
NLLR	0.140	0.125
OKAPI	0.161	0.146

When comparing the centralized baseline results with the merged results, we should take the second column, MAP@250, since the merged result lists are never contain more than 250 results.

5.3 Training Time and Merging Time

The time needed to train the SVM models was recorded, as was the time for merging the results using. Table 5.3 lists the time needed to train the best regression and preference models, and the time needed for merging 250 results, using either an SVM model or RR.

Table 5.3: Time needed to train a model and merge 250 results (including disk I/O time)

	Training time (in seconds)	Avg. merging time (in seconds)
Round Robin	-	0.8
Regression SVM	0.6	21
Preference SVM	150	28

Keep in mind that the listed training time is for that particular model only. The time needed to train and test all our models, in order to verify that some model was indeed the most robust model, was much more. Some models took over an hour to train, and we manually stopped a few models that seemingly did not converge as they were running for hours.

5.4 Important Features

The most robust model was a regression model, trained on all features but the third feature group. Since a linear kernel was used, the weights of these features could be easily extracted and are shown below in Table 5.4.

Table 5.4: Grouped and weighted feature list

1	+0.0033	1. Rank: (1000 – rank) / 1000 (range [0,1])
	−3.2926	2. Rank: rank (range [1…])
2	+2.1756	3. Cosine similarity query – title (range [0,1])
	+0.8261	4. Cosine similarity query – snippet (range [0,1])
	+0.1014	5. Cosine similarity query – URL (range [0,1])
3	n/a	6. Title: number of words (range [1…])
	n/a	7. Title: average word length in chars (range [1…])
4	+0.0837	8. Snippet: number of words (range [0…])
	−0.0386	9. Snippet: average word length in chars (range [1…])
5	−0.0988	10. URL: length in chars of FQDN (range [4…])
	−0.4186	11. URL: FQDN frequency in current list (range [0,1])
	+0.0196	12. URL: path-depth
	+1.1125	13. URL: average path length in chars (range [1…])
6	−0.1999	14. Query: number of words (range [1…])
	+0.0212	15. Query: average word length in chars (range [1…])
7	−0.5569	16. URL: contains tilde '~' (binary {0,1})
	−0.2965	17. URL: contains text 'home' (binary {0,1})
	+0.2500	18. Title: contains text 'home' (binary {0,1})
	−0.1353	19. LCS query – title (range [0,1])
	−0.2553	20. LCS query – snippet (range [0,1])
	−0.0012	21. LCS query – URL (range [0,1])
	−0.1630	22. LWO query – title (range [0,1])
	−0.1692	23. LWO query – snippet (range [0,1])
	−0.0012	24. LWO query – URL (range [0,1])
	−0.5258	25. LWO title – snippet (range [0,1])
	+0.9042	26. LWO title – URL (range [0,1])
	+0.1020	27. LWO URL – snippet (range [0,1])
8	+2.3412	28. Server usefulness: LAP (range [0,1])
	+0.2613	29. Server usefulness: GAP (range [0,1])
	+3.8140	30. Server usefulness: Merit (range [0…])

If the ranges of all features were normalized, then the most important features would be those with a high absolute weight. (A negative weight indicates that the feature should be penalized.) Since not all of our features are normalized, it is difficult to conclude which features are the most important ones.

5.5 RR & SVM Performance

This section reports the effects of the external influences on the merging performance. First, the effects of the selection policies will be illustrated in Figures 5.1 up to 5.6. Here, we explicitly show the number of search engines being merged, together with their combined amount of relevant pages, which is a direct result of the resource selection policy used.

As a short note, keep in mind that we want to select as few search engines as possible (e.g., to minimize network traffic and computing time) and at the same time, we want the merging performance to be as high as possible.

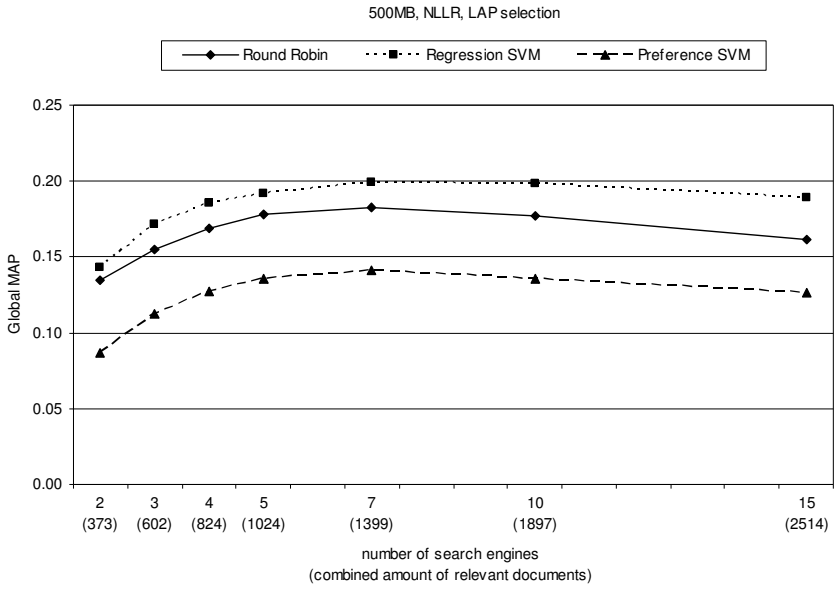

Figure 5.1: GMAP graphs of 3 merging methods, the search engines are selected from the 500MB NLLR split using the LAP selection policy

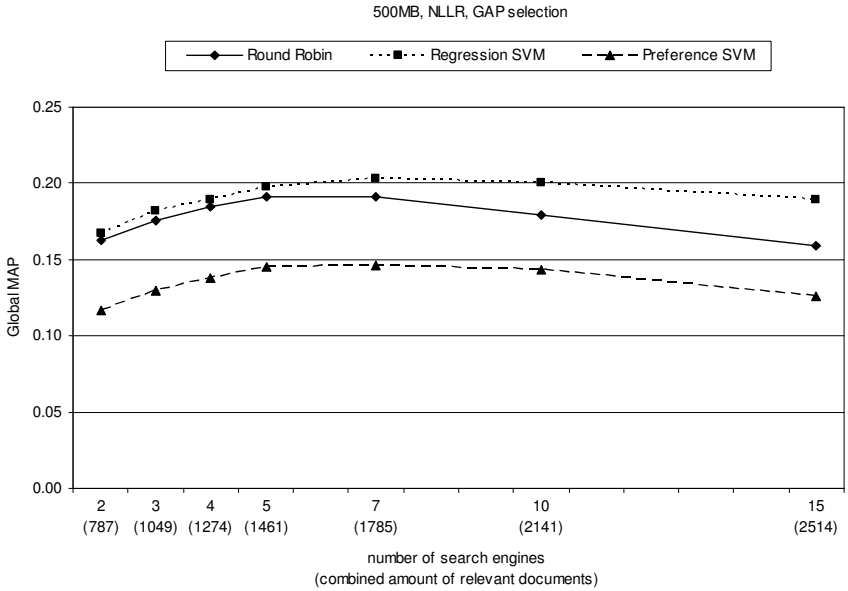

Figure 5.2: GMAP graphs of 3 merging methods, the search engines are selected from the 500MB NLLR split using the GAP selection policy

Figures 5.1 and 5.2 show that with the LAP or GAP selection policy, the regression approach is always better than our merging baseline, and that the preference approach is always worse. It should not be surprising that RR performs well, especially with GAP selection, since GAP is a near optimal selection policy for RR merging.

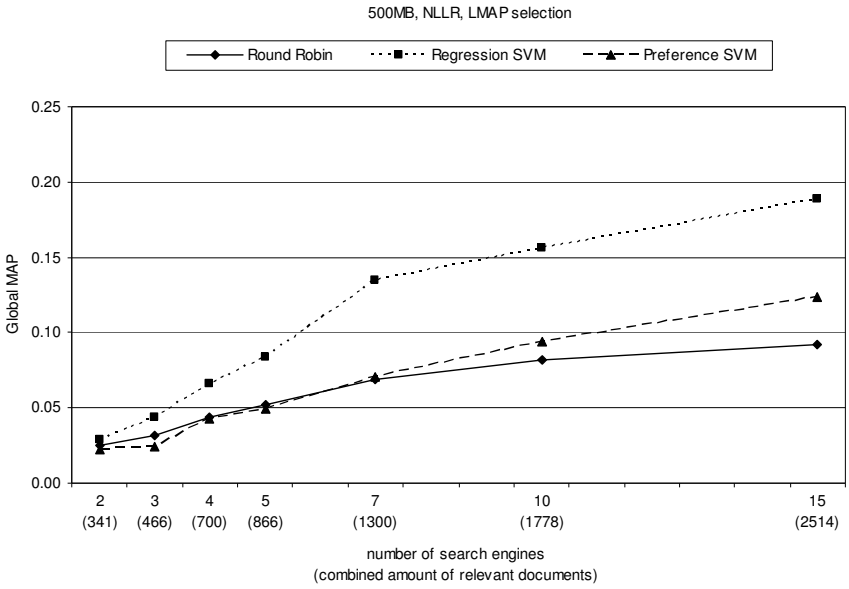

Figure 5.3: GMAP graphs of 3 merging methods, the search engines are selected from the 500MB NLLR split using the LMAP selection policy

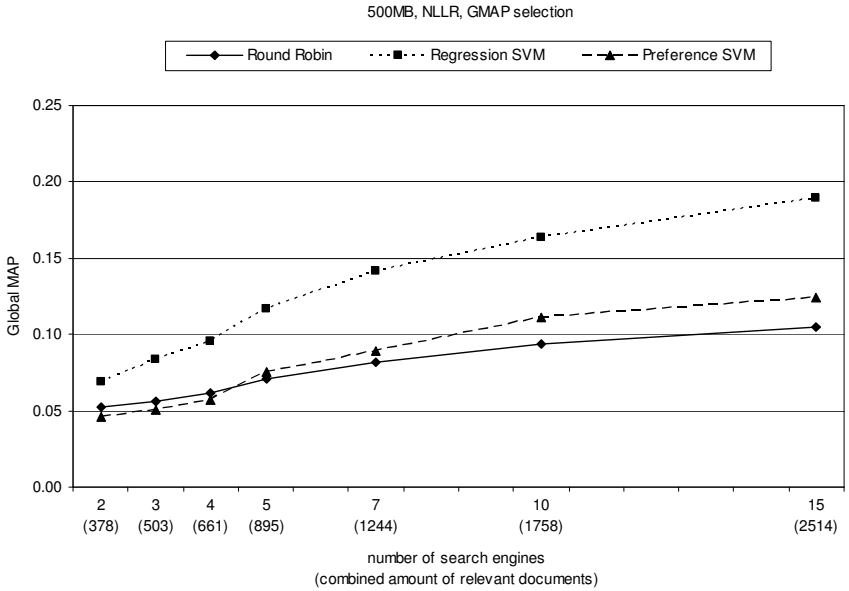

Figure 5.4: GMAP graphs of 3 merging methods, the search engines are selected from the 500MB NLLR split using the GMAP selection policy

Although LMAP and GMAP are performance measures, after comparing Figures 5.3 and 5.4 with the previous two, it is clear that selecting the same search engines for all queries is not a good idea. In other words, it is not wise to say, for example, that because Google is the best search engine on average, we will always use its results for result merging. The search engines should be selected on a per query basis, with the criterion that it has the best results for that query compared to the results of other search engines for that query.

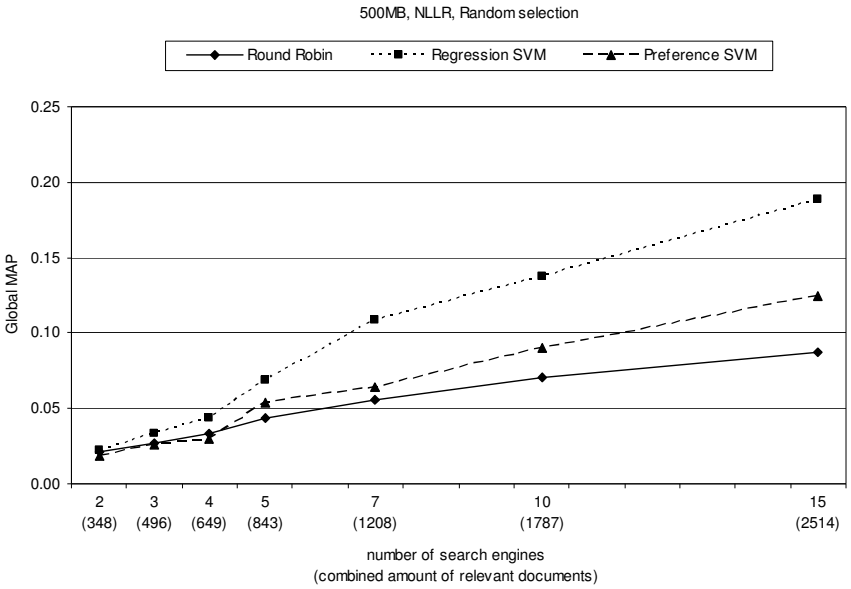

Figure 5.5: GMAP graphs of 3 merging methods, the search engines are selected from the 500MB NLLR split using the random selection policy

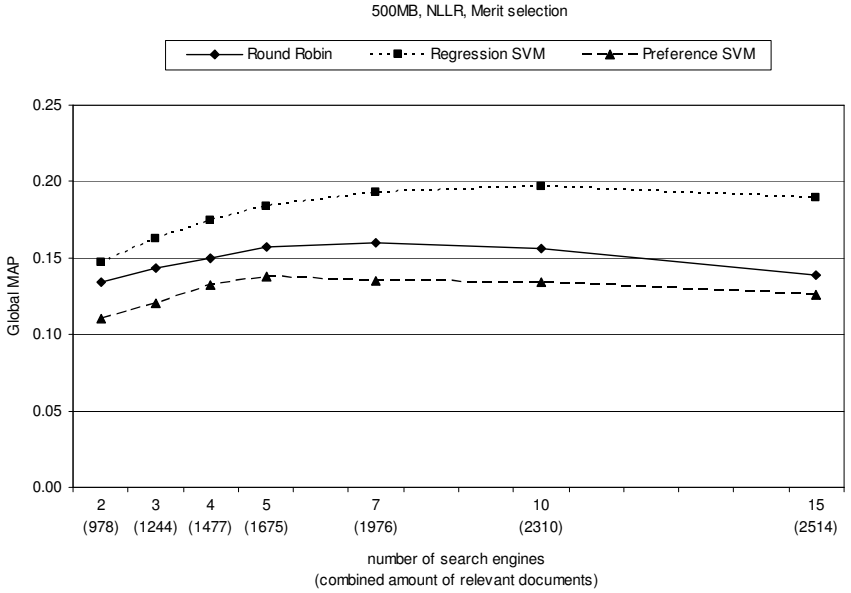

Figure 5.6: GMAP graphs of 3 merging methods, the search engines are selected from the 500MB NLLR split using the merit selection policy

The random and merit[1] selection policies are no performance measures. Figures 5.5 and 5.6 confirm that random selection is not desired and that merit selection works well, although not as good as LAP or GAP.

[1] A search engine's merit tells us about the number of relevant documents in its document collection for a given query. However, it does not say that the search engine actually retrieves them. As such, a search engine's merit is not a performance measure.

As can be seen from the previous graphs, the preference model was often (significantly) worse than both our baseline RR scheme and the regression model. Therefore, the next Figures, 5.7 up to 5.10, will only display the RR and the regression performances. We will use the term SVR to refer to our regression SVM model.

Together, Figures 5.1 up to 5.6 show the importance of the ordering of the search engines when doing RR merging. At the point where fifteen search engines are merged, all six selection policies selected the same fifteen engines, albeit in a different order. This difference in order dramatically affects RR merging, while both SVM approaches converge to certain value and seem to be more stable. Figure 5.7 clearly shows that when merging results of fifteen search engines, using any selection policy, the SVR graphs converge while the RR graphs do not.

Next, we will summarize the SVR and RR graphs for:
1. the 500MB collection split indexed with the NLLR IR- model; and,
2. the 100MB collection split indexed with the NLLR IR-model.

Figures 5.7 up to 5.10 plot the GMAP (or LMAP) against the number of search engines being merged, for a given selection policy and a given merging method. The names in the legend consist of: the name of the merging method (RR or SVR), a hash symbol '#', and the name of the selection policy.
For a given selection policy (for instance, one of the graphs in Figures 5.7 up to 5.10) and the number of search engines to merge, the combined number of relevant documents are shown separately in Tables 5.5 and 5.6.

Table 5.5: Combined number of relevant pages of the search engines, selected from the 500MB NLLR collection with the given selection policy.

	2	3	4	5	7	10	15
LAP	373	602	824	1024	1399	1897	2514
GAP	787	1049	1274	1461	1785	2141	2514
LMAP	341	466	700	866	1300	1778	2514
GMAP	378	503	661	895	1244	1758	2514
RANDOM	348	496	649	843	1208	1787	2514
MERIT	978	1244	1477	1675	1976	2310	2514

Table 5.6: Combined number of relevant pages of the search engines, selected from the 100MB NLLR collection with the given selection policy.

	2	3	4	5	7	10	15
LAP	150	219	307	388	550	817	1177
GAP	564	688	817	940	1107	1350	1658
LMAP	80	138	175	213	279	407	569
GMAP	96	113	145	183	260	364	562
RANDOM	48	72	120	149	218	301	507
MERIT	699	888	1042	1170	1384	1621	1917

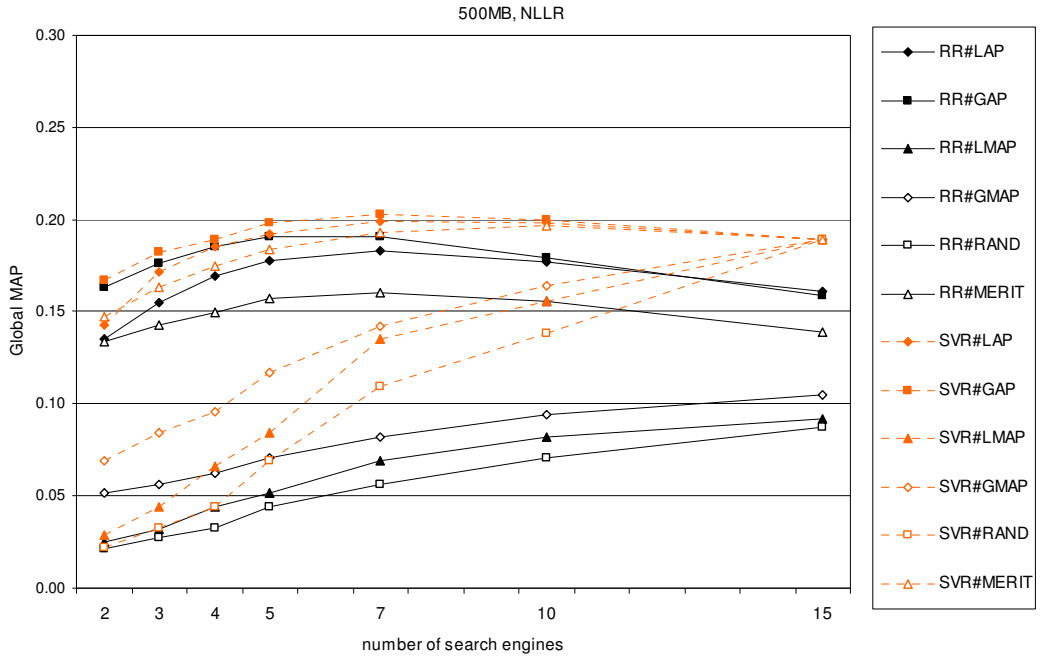

Figure 5.7: GMAP of tests on the 500MB NLLR split.

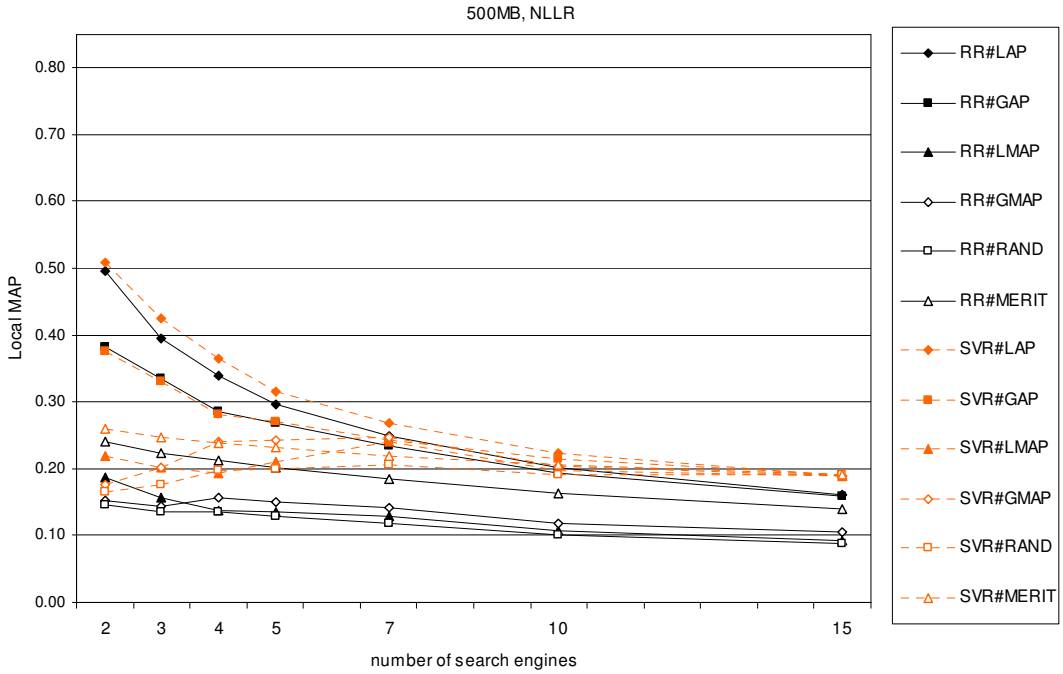

Figure 5.8: LMAP of tests on the 500MB NLLR split.

The effects of the selection policies are clearly visible in Figures 5.7 and 5.8. The desired policies, in order of preference, are GAP, LAP, and merit. The SVR performance is significantly better ($p<0.05$) than RR in almost all (500MB, NLLR) experiments; only the LAP and GAP points for less than 15 search engines did not prove significant.

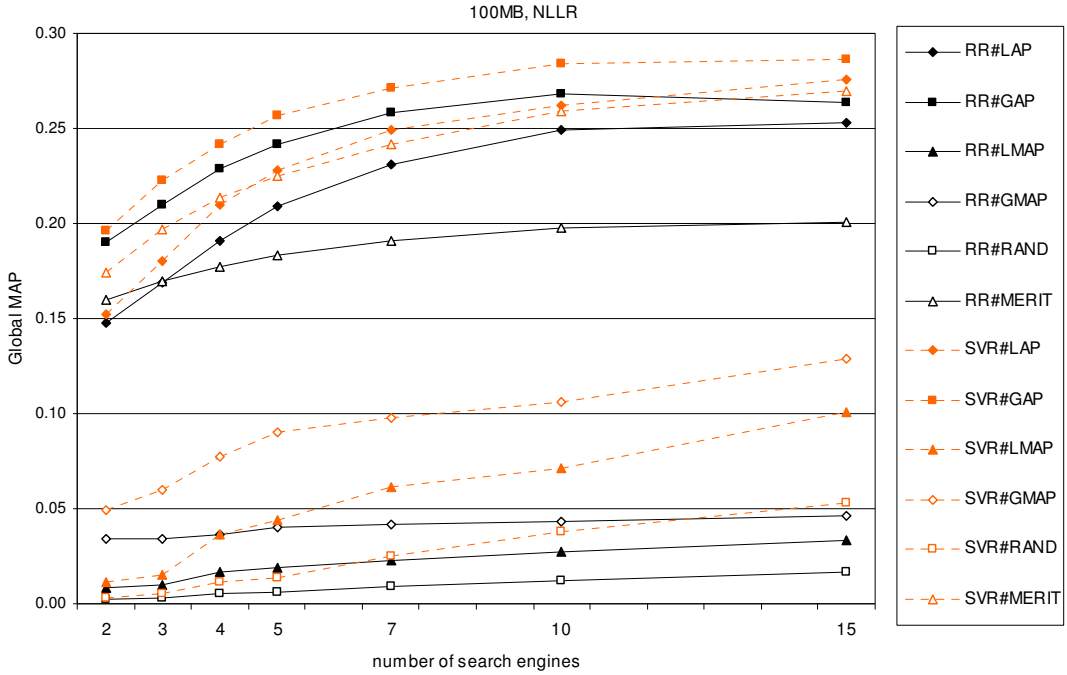

Figure 5.9: GMAP of tests on the 100MB NLLR split

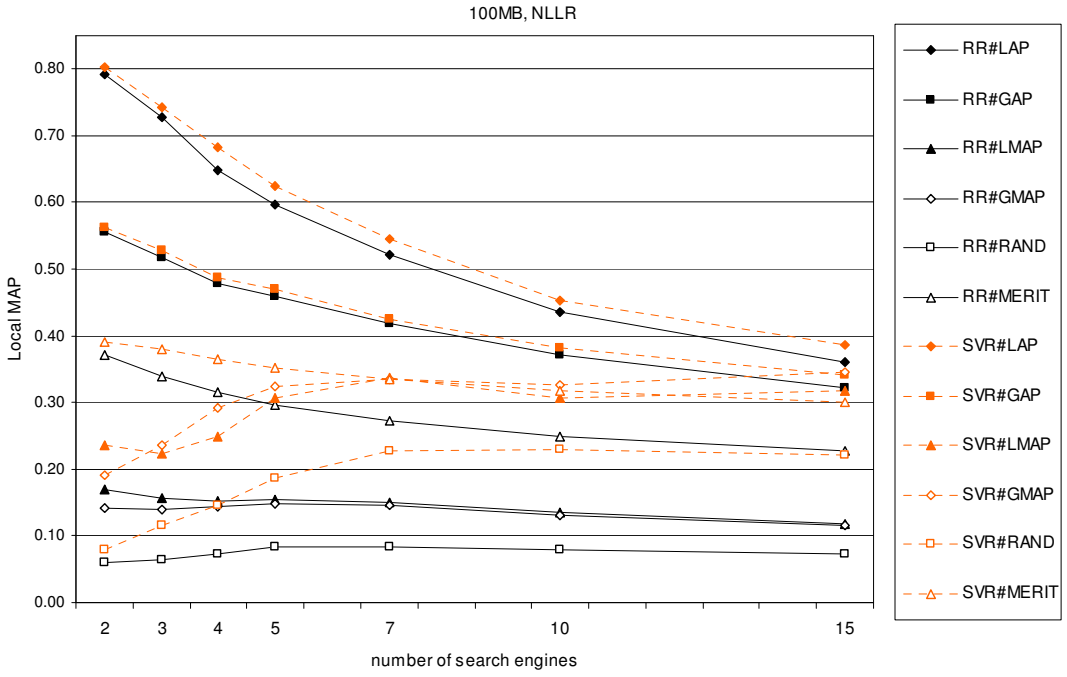

Figure 5.10: LMAP of tests on the 100MB NLLR split

Here, in Figures 5.9 and 5.10, the SVR performance is significantly better ($p<0.05$) than RR in all (100MB, NLLR) experiments, except for the experiments with the LAP and GAP selection policies.

The same effects can be seen with the OKAPI IR-model; therefore, it is of no additional value to display those tables and figures. However, since OKAPI produces slightly better results, the merged results are also slightly better.

The astute reader might ask why, as the number of search engines increases, the regression performance in the 500MB collection split decreases at some point. This is because we only merged 250 results and each search engines contributes an equal amount of results to this merged list. Therefore, if we merge the results of fifteen engines, each engine contributes (its first) 16.7 results on average, and we miss out quite a few relevant results, since relevant results are often found after the 16th result.

In Section 4.1.2, we assumed, besides for efficiency reasons, that it would not be a good idea to merge the results of all search engines. We assumed that many engines would not contribute any relevant results and therefore, the merging performance would degrade. We also assumed that a centralized IR system would perform at least as good as a Distributed IR system when merging the results of all search engines.
We conducted a small additional experiment using the SVR model to merge all seventy-nine search engines of the 100MB NLLR split; the GMAP was 0.234. The GMAP when merging all engines of the 500MB NLLR split using the SVR model was 0.189. Our centralized baseline had a MAP@250 of 0.125 (see Table 5.2). These experiments invalidate our assumption for not merging the results of all search engines (of course, this is not efficient).
These experiments show that the performance of a Distributed IR system can be much better than that of a centralized IR system in many scenarios; for the SVR merging method it does not matter which selection policy is used, and even RR merging, with the right selection policy, is much better than the centralized system.

Chapter 6

Discussion

6.1 More on the Results

The only difference in Figures 5.7 and 5.9 is the document collection size of the search engines, and as a direct result, the 100MB split has more collections.

The difference in collection size affects the performance of the merging methods in several ways. First, the SVR graphs do not converge just as they did in Figure 5.7. This is because no selection policy selects the same fifteen search engines. Second, the merging methods yield much higher performance in the 100MB split than in the 500MB split.

The question arises why this is the case. It could be attributed to: one, a more fine-grained resource selection, as there are more search engines to select from; two, IR-models may work better on smaller collections thereby producing better result pages; or three, a combination of both one and two.

After inspecting the selected search engines in the 100MB split with the highest GAP, we noted that many had an LAP of either one, or close to one, meaning that those engines ranked all or almost all of their relevant documents in the top. This is not the case in the 500MB split. Thus, we can conclude that the better merging is a direct result of better result pages.

6.2 Kernels and Overfitting

A large part of this work was devoted to Support Vector Machine (SVM) related material; however, it was never the intention to squeeze the last bit of performance out of the SVM models. We did put some effort in selecting appropriate features and in optimizing the SVM parameters.

We did not present all results of the preliminary tests; we only concluded which data chunks and features were promising candidates for further optimizing the SVM model.

Nevertheless, all preliminary tests using the Radial Basis Function (RBF) kernel consistently proved much better than the tests using the linear kernel when evaluated on the training set, and proved worse when evaluated on the test set. We have no explanation for this apparent overtraining that occurs with the RBF kernel. However, according to Keerthi [19], every linear kernel can be expressed by a RBF kernel with the right parameters for γ and C. Therefore, we can conclude that the optimal parameters for the RBF kernel were not found.

The fact that the linear kernel works best tells us that the data can be separated "well" in input space.

6.3 Similar Regression and RR Behavior

In the experiments with the GAP or LAP selection policy, the regression model behaves similarly to the RR-merge, although always yielding slightly better results.
One possible explanation is that this similar behavior just happened by chance. Another explanation is that the regression model was affected by RR, since the training was done after a RR-merge of the training samples: when we trained on data from multiple search engines, we first did an RR merge of their results.
The RR-merge only affected the first feature, namely the rank ratio, which is (1000 minus the result's rank - assigned by RR) divided by 1000. The rest of the features, and the target values, remained the same. In addition, if we look at the learned weight for the first feature of the regression model, we see that it is almost zero, indicating that the regression model practically ignores the effects of RR merging. Finally, this similar behavior only occurs with the LAP and GAP selection policies. Therefore, we conclude that the similar behavior occurred by chance.

The fact that RR works well, in combination with the LAP and GAP policies, indicates that a result's rank and the "usefulness" of the search engine are the most important features. The learned weights of the regression model (see Table 5.4) seem to confirm that the rank and "usefulness" are the most important features.

6.4 Efficient Result Set Selection

GAP proved to be the best resource selection policy, better than the merit and LAP policy. However, the learned weight of the GAP feature is much smaller than the learned weights of the merit and LAP features. We suspect that the optimal selection strategy might be based on a combination of the merit and LAP features.
An LAP of one means that all relevant documents in the search engine's document collection were retrieved and placed in the first N ranks. It does not say anything about a number of relevant documents retrieved. Furthermore, you cannot compare the LAPs of two search engines to see which one retrieved more results that are relevant. This is where GAP has an advantage: you can compare the GAPs of two engines to see which of the two retrieved more results that are relevant. Still, both GAP and LAP do not say anything about the number of results actually retrieved.
An engine's merit does not say anything about a number of documents retrieved either.

However, if, for a given query, an engine's LAP is one and its merit is M, then we know for sure that the first M results in its result list are relevant. Actually, with an engine's merit and LAP, we can calculate the probability $P_i(r \mid R)$, where r is the number of relevant documents retrieved when downloading R documents from some server i.
Viewing the problem of result merging as that of re-ranking a set of results, much efficiency can be gained by selecting the right set of results to begin with. With the merit and LAP statistics, we can not only select the most promising search engines, but we can also decide on the amount of results to retrieve from each engine. We believe that this is the optimal selection policy: it enables us to minimize the amount of data traffic while we maximize the amount of relevant documents in the result set.

6.5 LMAP

We devised LMAP to allow easy comparison between the merging-performance of different Distributed IR systems. Our experiments show that LMAP does not fit this task: the effects of the selection policy are too strong and the LMAP values vary too much.
Hypothetically speaking, if some merging method A is always better than merging method B, then, this should be apparent from the LMAP figures no matter what resource selection policy is used: all values of A should be higher than all values of B.
Perhaps if the LMAP were computed based on the amount of relevant results in the combined result lists (instead of in the combined document collection of the search engines), it would fit the task of easy comparison of different merging systems.

6.6 Preference-SVM

Joachims [18] reported on the successful improvements of his SVM algorithm in a metasearch scenario using clickthrough data. Our experiments, using TREC relevance judgments as the "source of clickthrough data", indicate that preference-SVM may not be suited for this kind of data. The work of Verberne et al. [40] resembles our research in some way: they also used a preference-SVM and a (logistic) regression approach and had similar conclusions. Their results with preference-SVM are very poor compared to the results obtained with logistic regression or genetic algorithms.

Chapter 7

Conclusion

This research aimed at improving the efficiency and the performance of result merging, one of three major problems in Distributed Information Retrieval. We gained efficiency by only allowing result lists, and not actual documents, to be downloaded at query-time. With the given restriction, we identified three methods that potentially improve current result merging strategies. Two methods are based on Support Vector Machine (SVM) learning: preference-SVM, and regression-SVM. The third one is Round Robin and functions as our merging baseline. We implemented these models in a distributed environment, which we had to build ourselves before we could train and test the models. In the following paragraphs, we repeat our research questions and answer them accordingly.

Q1. Which of RR-merging, preference-SVM, and regression-SVM is recommended and why?

A1. Our results show that regression-SVM is always as good as, or better than, RR merging, whereas preference-SVM is significantly worse than both models most of the times. However, RR merging requires fewer operations and it is an order of magnitude faster. We also saw that the selection policy had a great impact on both RR and on regression SVM; however, the impact on regression-SVM grows smaller as more search engines are selected.
We recommend using RR merging only in cases where the following conditions apply: first, the primary concern is the speed of the system, which must be as high as possible; second, it is certain that the selection policy is good enough.
We recommend using regression-SVM in all other cases.

Q2. Using information only from result pages and the broker's selection mechanism, what are suitable information sources to use for result merging, and what are their weights?

A2. The most robust model in our experiments was a regression SVM model with a linear kernel. The features used to train this model are shown in Table 5.4, along with their learned weights. If the ranges of all features were normalized, then the most important ones would be those with the highest absolute weights. Since not all our features are normalized, it is difficult to say which features are most important. However, as discussed in Section 6.4, we believe that the rank and server usefulness can be considered as the most important features.

Q3. How vulnerable are the merging strategies to external influences like the number of result lists to merge, or the quality of the result pages?

A3. The most influential factors on the merging performance are in the first place the resource selection policy, and in the second place the number of search engines to merge. The selection policy selects a set of search engines (of a certain quality), and orders them.

The ordering of this set affects only RR and has no effect on regression-SVM. Summarizing, all merging strategies are vulnerable to the selection policy. However, as the number of selected search engines increases, the effect of the selection policy on the SVM methods decreases.

Q4. How well do these result merging strategies perform in terms of the cpu-time / performance (tp) ratio?

A4. The regression model's performance is often better than that of RR. The preference model's performance is often worse than RR. Ignoring the cpu-time for training the SVM models, the cpu-time of the SVM approaches, for doing result merging, is an order of magnitude larger than that of RR.

For instance, in the case where the difference between RR and regression SVM is maximal (the random selection policy, at fifteen peers), the tp-ratio of RR is 11, whereas the tp-ratio of the regression-SVM is 132. In the case where the difference between RR and SVM is minimal (the random selection policy, at two peers), RR's tp-ratio is 38, whereas the SVM's tp-ratio is 454.

Chapter 8

Future Work

This research sought suitable features for training SVM models for the task of result merging, while only extracting these features from the result pages and from the broker's resource selection component (for reasons of efficiency and trust: we assumed that we were operating in an uncooperative environment).
These models were tested in different scenarios: one, where 79 search engines had a document collection of roughly 100MB; and two, where 15 engines had a collection of roughly 500MB. Furthermore, two different IR-models (NLLR and OKAPI) were used in each scenario, yielding four different test beds.
However, in Distributed Web search, it is likely that: many search engines will have some collection overlap, that their collection size may vary widely, that there is more variety in the IR-models used, that there are topical (expert) databases, and that clusters of cooperative search engines will emerge to gain advantage over the competition.

If we assume cooperative and truly heterogeneous environments, then the merging component (e.g., SVM models) can be trained on many more features. For instance, models could be trained on the following features: the collection size, collection overlap, the search engines' main topics, the IR-model, the document score, and much more.
With collection-overlap comes the additional task of duplicate detection and deletion. More generally, look-a-like detection can be very important, even in cases without collection overlap, since this allows the search engine to decide whether to present as many distinct documents as possible, or as many look-a-likes as possible.

Finally, based on our findings, we believe that much performance gain can be obtained by allowing the best search engines to contribute more results in the merged list. Another thing we noticed is that sometimes a hint can be found hidden in (concatenated) words, such as "http://www.somethingyouarelookingfor.com/whatis.html", which points towards Natural Language Processing (NLP) techniques, as a logical next step.

Bibliography

[1] J. A. Aslam, and M. Montague, "Models for metasearch," in Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, New Orleans, Louisiana, United States, 2001.

[2] R. A. Baeza-Yates, and B. Ribeiro-Neto, *Modern Information Retrieval*: Addison-Wesley Longman Publishing Co., Inc., 1999.

[3] R. A. Baeza-Yates, C. Castillo, F. Junqueira *et al.*, "Challenges on Distributed Web Retrieval," in Proceedings of the 23rd International Conference on Data Engineering, The Marmara Hotel, Istanbul, Turkey, 2007.

[4] P. Bailey, N. Craswell, and D. Hawking, "Engineering a multi-purpose test collection for web retrieval experiments," *Inf. Process. Manage.,* vol. 39, no. 6, pp. 853-871, 2003.

[5] B. T. Bartell, G. W. Cottrell, and R. K. Belew, "Automatic combination of multiple ranked retrieval systems," in Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval, Dublin, Ireland, 1994.

[6] N. J. Belkin, P. Kantor, E. A. Fox *et al.*, "Combining the evidence of multiple query representations for information retrieval," *Information Processing & Management,* vol. 31, no. 3, pp. 431-448, 1995.

[7] M. K. Bergman, "The Deep Web: Surfacing Hidden Value," *Journal of Electronic Publishing,* vol. 7, no. 1, 2001.

[8] C. J. C. Burges, "A Tutorial on Support Vector Machines for Pattern Recognition," *Data Min. Knowl. Discov.,* vol. 2, no. 2, pp. 121-167, 1998.

[9] J. Callan, "Distributed information retrieval," *Advances in information retrieval*, pp. 127-150, 2000.

[10] J. Callan, and M. Connell, "Query-based sampling of text databases," *ACM Trans. Inf. Syst.,* vol. 19, no. 2, pp. 97-130, 2001.

[11] J. P. Callan, Z. Lu, and W. B. Croft, "Searching distributed collections with inference networks," in Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval, Seattle, Washington, United States, 1995.

[12] G. Cao, J.-Y. Nie, and J. Bai, "Integrating word relationships into language models," in Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, Salvador, Brazil, 2005.

[13] Y. Cao, J. Xu, T.-Y. Liu *et al.*, "Adapting ranking SVM to document retrieval," in Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, Seattle, Washington, USA, 2006.

[14] C. Cortes, and V. Vapnik, "Support-Vector Networks," *Mach. Learn.,* vol. 20, no. 3, pp. 273-297, 1995.

[15] M. Farah, and D. Vanderpooten, “An outranking approach for information retrieval,” *Information Retrieval,* vol. 11, no. 4, pp. 315-334, Saturday, February 16, 2008, 2008.

[16] R. Herbrich, T. Graepel, and K. Obermayer, "Large margin rank boundaries for ordinal regression," *Advances in Large Margin Classifiers*, pp. 115-132: MIT Press, Cambridge, MA, 2000.

[17] T. Joachims, "Making large-Scale SVM Learning Practical," *Advances in Kernel Methods - Support Vector Learning*, B. Schölkopf, C. Burges and A. Smola, eds.: MIT-Press, 1999.

[18] T. Joachims, “Optimizing search engines using clickthrough data,” in Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, Edmonton, Alberta, Canada, 2002.

[19] S. S. Keerthi, and C.-J. Lin, “Asymptotic behaviors of support vector machines with Gaussian kernel,” *Neural Comput.,* vol. 15, no. 7, pp. 1667-1689, 2003.

[20] V. Lavrenko, and W. B. Croft, “Relevance based language models,” in Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval, New Orleans, Louisiana, United States, 2001.

[21] S. Lawrence, and C. L. Giles, “Inquirus, the NECI meta search engine,” in Proceedings of the seventh international conference on World Wide Web 7, Brisbane, Australia, 1998.

[22] S. Lawrence, and C. L. Giles, “Accessibility of information on the Web,” *Intelligence,* vol. 11, no. 1, pp. 32-39, 2000.

[23] J. Madhavan, S. R. Jeffery, S. Cohen *et al.*, “Web-scale Data Integration: You can only afford to Pay As You Go,” in CIDR, 2007.

[24] R. Nallapati, “Discriminative models for information retrieval,” in Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, Sheffield, United Kingdom, 2004.

[25] L. Page, S. Brin, R. Motwani *et al.*, *The PageRank Citation Ranking: Bringing Order to the Web*, Tech. report, Stanford University, 1999.

[26] G. Paltoglou, M. Salampasis, and M. Satratzemi, "Results Merging Algorithm Using Multiple Regression Models," *Advances in Information Retrieval*, Lecture Notes in Computer Science, pp. 173-184: Springer Berlin / Heidelberg, 2007.

[27] I. Podnar, M. Rajman, T. Luu *et al.*, "Scalable Peer-to-Peer Web Retrieval with Highly Discriminative Keys." pp. 1096-1105.

[28] J. M. Ponte, and W. B. Croft, “A language modeling approach to information retrieval,” in Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, Melbourne, Australia, 1998.

[29] Y. Rasolofo, D. Hawking, and J. Savoy, “Result merging strategies for a current news metasearcher,” *Inf. Process. Manage.,* vol. 39, no. 4, pp. 581-609, 2003.

[30] S. E. Robertson, S. Walker, S. Jones *et al.*, "Okapi at TREC-3." pp. 109-26.

[31] G. Salton, A. Wong, and C. S. Yang, “A vector space model for automatic indexing,” *Commun. ACM,* vol. 18, no. 11, pp. 613-620, 1975.

[32] M. Shokouhi, "Segmentation of Search Engine Results for Effective Data-Fusion," *Advances in Information Retrieval*, Lecture Notes in Computer Science, pp. 185-197: Springer Berlin / Heidelberg, 2007.

[33] L. Si, and J. Callan, “Using sampled data and regression to merge search engine results,” in Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval, Tampere, Finland, 2002.

[34] L. Si, and J. Callan, “A semisupervised learning method to merge search engine results,” *ACM Trans. Inf. Syst.,* vol. 21, no. 4, pp. 457-491, 2003.

[35] L. Si, and J. Callan, “Modeling search engine effectiveness for federated search,” in Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, Salvador, Brazil, 2005.

[36] G. Skobeltsyn, T. Luu, I. P. Zarko *et al.*, “Web text retrieval with a P2P query-driven index,” in Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, Amsterdam, The Netherlands, 2007.

[37] M. D. Smucker, J. Allan, and B. Carterette, “A comparison of statistical significance tests for information retrieval evaluation,” in Proceedings of the sixteenth ACM conference on Conference on information and knowledge management, Lisbon, Portugal, 2007.

[38] C. Tang, Z. Xu, and S. Dwarkadas, “Peer-to-peer information retrieval using self-organizing semantic overlay networks,” in Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications, Karlsruhe, Germany, 2003.

[39] V. N. Vapnik, *The nature of statistical learning theory*: Springer-Verlag New York, Inc., 1995.

[40] S. Verberne, S. Raaijmakers, D. Theijssen *et al.*, “Learning to Rank Answers to Why-Questions,” in 9th Dutch-Belgian Information Retrieval Workshop (DIR 2009), Enschede, 2009, pp. 34-41.

[41] E. Voorhees, D. K. Harman, and T. National Institute of Standards and, "TREC : experiment and evaluation in information retrieval," *Digital libraries and electronic publishing*.

[42] E. M. Voorhees, N. K. Gupta, and B. Johnson-Laird, "The Collection Fusion Problem." pp. 500-225.

[43] P. Wellens, “Een Introductie tot Support Vector Machines,” master thesis, Wiskunde en Informatica, Universiteit Antwerpen, 2005.

Appendix

A. Pseudo code for LCS

The pseudocode for finding the Longest Common Substring in two strings.

```
INITIALIZE A to the first string
INITIALIZE B to the second string
INITIALIZE LCS to zero

REMOVE all non alpha-numerical characters from A and B

SWAP A and B IF A is longer than B

FOREACH word_a in A

   FOREACH word_b in B

      INITIALIZE word_a' to word_a
      INITIALIZE word_b' to word_b
      INITIALIZE len to zero

      WHILE word_a' is word_b'

         INCREASE len by one

         IF next words exists in A and B THEN
            word_a' ← next word after word_a' in A
            word_b' ← next word after word_b' in B
         ELSE
            BREAK loop
         ENDIF

      ENDWHILE

      IF len is greater than LCS THEN
         LCS ← len
      ENDIF

   ENDFOR

ENDFOR

RETURN LCS divided by the amount of words in A
```

B. Pseudo code for LWO

The pseudocode for finding the Longest Word Order in two strings.

```
INITIALIZE A to the first string
INITIALIZE B to the second string
INITIALIZE LWO to zero

REMOVE all non alpha-numerical characters from A and B

SWAP A and B IF A is longer than B

FOREACH word_a in A

   INITIALIZE len to zero
   INITIALIZE word_a' to word_a

   FOREACH word_b in B

      IF word_a' is word_b THEN

         INCREASE len by one

         IF there is a next word after word_a' in A THEN
            word_a' ← next word after word_a' in A
         ELSE
            BREAK loop
         ENDIF

      ENDIF

   ENDFOR

   IF len is greater than LWO THEN
      LWO ← len
   ENDIF

ENDFOR

RETURN LWO divided by the amount of words in A
```

C. Pseudo code for Labeling Preference Pair Constraints

```
INITIALIZE Q to the odd queries used for training
INITIALIZE C to the selected search engines
INITIALIZE qid to one
INITIALIZE training to the empty collection

FOREACH q in Q

   INITIALIZE results to the first 250 results of **RR(q, C)

   FOREACH r in results

      IF r is not irrelevant THEN

         ADD (1 qid:qid feature-vector of r) to training

         FOREACH r' above r in results

            IF r' is less relevant than r THEN

               ADD (0 qid:qid feature-vector of r') to training

            ENDIF

         ENDFOR

         ADD 30 random constraints to training

         INCREASE qid by one

      ENDIF

   ENDFOR

ENDFOR

RETURN training

**RR(q, C) is the Round Robin merge of the
  results of each collection in C for query q.
```

Joachims added fifty random constraints from his candidate set for each clicked result. This candidate set, the combined set of results from 5 search engines, consisted of 500 results. Since we only have 250 results in our candidate set, we decided to lower the number of additional constraints to 30.