NORTHEASTERN UNIVERSITY
GRADUATE SCHOOL OF COMPUTER SCIENCE
Ph.D. THESIS APPROVAL FORM

THESIS TITLE: Collection Construction Methodologies for Learning to Rank

AUTHOR: Stefan Savev

Ph.D. Thesis Approved to complete all degree requirements for the Ph.D. Degree in Computer Science.

Thesis Advisor

Date 12/18/12

Thesis Reader

Date 7/7/2012

Thesis Reader

Date 9/10/2012

Thesis Reader

Date 9/7/2012

GRADUATE SCHOOL APPROVAL:

Date 12/18/12

Director, Graduate School

COPY RECEIVED IN GRADUATE SCHOOL OFFICE:

Recipient's Signature

Date 12/18/2012

Distribution: Once completed, this form should be scanned and attached to the front of the electronic dissertation document (Page 1). An electronic version of the document can then be uploaded to the Northeastern University-UMI website.
Collection Construction Methodologies for Learning to Rank

A dissertation presented
by
Stefan Savev
to
The College of Computer and Information Science
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in the subject of
Computer Science
Northeastern University
Boston, Massachusetts
December 2012
Collection Construction Methodologies for Learning to Rank

Abstract

Ranking documents in response to user queries is one of the fundamental problems in Information Retrieval. Learning to Rank has emerged as an effective approach for data-driven construction of ranking algorithms. Although many algorithms have been created, the effect of the properties of the training data through which such algorithms are developed has not been systematically studied. The creation of a learning dataset requires great effort since every document in the dataset has to be manually labeled with its degree of relevance. Motivated by the importance of efficiently creating quality datasets, I study 1) the effect of characteristics of the training dataset on algorithm quality and 2) theoretically founded methods for construction of training datasets. With regard to the first topic, I establish through a number of controlled experiments that properties such as the distribution of documents across relevance grades and distances between relevance categories are useful predictors of dataset quality. Second, drawing on the statistical theory of Optimal Design of Experiments I provide a theoretical foundation for the identified characteristics in training set selection criteria. Its underlying intuition is that one should simultaneously maximize the diversity between feature vectors and the representation of relevant documents.
Contents

Title Page ................................................................. i
Abstract ................................................................. iii
Table of Contents ....................................................... iv
List of Figures .......................................................... viii
List of Tables ........................................................... x
Acknowledgments ......................................................... xi

1 Introduction .......................................................... 1
  1.1 Background Information ........................................... 3
    1.1.1 Search engine architecture ................................ 3
    1.1.2 Information Retrieval ....................................... 3
  1.2 Learning to Rank ................................................... 5
    1.2.1 Test collections ............................................. 6
    1.2.2 Relevance judgments ....................................... 8
    1.2.3 Evaluation ................................................ 9
    1.2.4 Training collections ...................................... 10
  1.3 Related Work ..................................................... 13
  1.4 Experimental Data ............................................... 14
  1.5 Research Questions .............................................. 15

2 Information Retrieval Methods and Learning to Rank Algorithms 18
  2.1 Information Retrieval Models .................................... 19
    2.1.1 Early models .............................................. 19
    2.1.2 Text-matching models based on probability ................. 19
  2.2 Learning Algorithms ............................................ 22
    2.2.1 Linear regression ......................................... 24
    2.2.2 Logistic regression ....................................... 25
    2.2.3 Neural networks .......................................... 26
2.2.4 Boosting ................................................. 27
2.2.5 Support vector machine .............................. 27
2.2.6 Learning to rank algorithms .......................... 29
2.2.7 Comparison of RankNet, Rank Boost and Ranking SVM .... 36

2.3 Classification of Learning Algorithms ..................... 39

3 The Theory of Optimal Design of Experiments .......... 40
3.1 Introduction .............................................. 40
3.2 Optimal Design of Experiments for the Linear Model .......... 40
3.3 Illustration of D-optimal Designs ........................ 46
3.4 Relationship of ODE to Generalization Error ............... 47
3.5 Optimization Algorithm .................................. 48
3.6 Non-linear Models ........................................ 49
3.6.1 Logistic regression .................................... 49
3.6.2 Neural networks ....................................... 52
3.6.3 Gaussian processes ................................... 54
3.7 Active Learning .......................................... 56
3.7.1 Active learning for learning to rank ...................... 58
3.8 Unsuitability of Active Learning for Training Collection Construction 61
3.8.1 Practical evidence ..................................... 61
3.8.2 Theoretical arguments ................................ 62

4 Empirical Studies of the Properties of Datasets for Learning to Rank .......... 64
4.1 Effect of the Distribution of Labels across Different Grades of Relevance in the Training Set .... 64
4.2 Effect of the Distribution of Feature Values on the Quality of the Training Set .................... 67
4.2.1 Experiment design ..................................... 73
4.2.2 Analysis .............................................. 74
4.3 Interaction between Feature Values and Relevance Categories ........ 76
4.4 Effect of Number of Documents per Query .................... 80
4.4.1 Experiment “Variable Number of Documents per Query” .... 80
4.4.2 Experiment “Number of Documents per Query for Different Selection Strategies” ........................................ 81
4.5 Conclusion ............................................................................................................ 83

5 Application of the Theory of Optimal Design of Experiments to Construction of Training Collection for Learning to Rank .................................................. 86
5.1 Introduction .......................................................................................................... 86
5.2 Related Work ....................................................................................................... 87
5.3 Experimental Design .......................................................................................... 88
   5.3.1 Datasets .......................................................................................................... 88
   5.3.2 Learning algorithms ....................................................................................... 91
5.4 Document Selection Procedures ........................................................................ 93
   5.4.1 Selection of random documents (“Random”) .............................................. 95
   5.4.2 Selection of top-ranked documents (“Top by score”) ............................... 97
   5.4.3 A mixture of random documents and top-ranked documents (“Random + top by score”) ........................................ 97
   5.4.4 Selection based on depth pooling (“Depth pool”) ....................................... 97
   5.4.5 Selection based on StatAP (“StatAP”) ........................................................ 98
   5.4.6 Selection of documents with high BM25 score (“top by BM25”) .......... 98
   5.4.7 Selection of random documents and documents with high BM25 score (“Random + top by BM25”) .................. 98
   5.4.8 Selection of documents based on ODE without top documents (“ODE”) .... 99
   5.4.9 Selection of documents based on ODE with top-ranked documents (“ODE + top”) ........................................ 99
   5.4.10 Selection of documents based on projection of documents on a random line (“Random projection”) .................. 100
5.5 Results and Analysis ............................................................................................ 102
   5.5.1 Methods that do not select explicitly top-ranked documents ................. 107
   5.5.2 Methods that explicitly select top-ranked documents ............................ 107
   5.5.3 Use of the BM25 score instead of the relevance score ............................ 108
   5.5.4 Interaction between the data selection methods, learning algorithms and dataset sizes .................................................. 109
6 Conclusion

6.1 Training Datasets Independent of the Learning to Rank Algorithm . 119
6.2 Difference between Training Collections and Test Collections . . . . 120
## List of Figures

1.1 Experimental setup for learning to rank ........................................... 4
3.1 Illustration of D-optimal design ......................................................... 46
3.2 Effect of $\lambda$ in D-optimal design .................................................. 47
3.3 Adaptive D-optimal design for logistic regression ............................... 52
4.1 Analysis of learning algorithm performance based on the distribution of relevance categories in the training dataset ....................... 68
4.2 Illustration of feature distribution of documents by relevance grade ...... 71
4.3 Illustration of the effect of distances between the feature vectors of two relevance grades .................................................. 72
4.4 Illustration of the experimental setup to observe the effect of the distance between the distributions of feature vectors from different relevance grades .................................................. 74
4.5 Effect of the distance between documents from one relevance category and non-relevant documents on the dataset quality ............ 77
4.6 Comparison of the effect of different relevance categories on the dataset quality .................................................. 79
4.7 Effect of the number of documents per query over dataset quality ....... 82
4.8 Mean and variance of NDCG@10 over all learning to rank algorithms as a function of the number of documents per query ............... 83
4.9 Effect of the number of documents per query for datasets consisting of a mixture of top-ranked and random documents ............... 84
4.10 Effect of two extreme values of number of documents per query for datasets consisting of a mixture of top-ranked and random documents 85
5.1 Distribution of the number of documents per query in MSLR-WEB30K 90
5.2 Illustration of the experimental design for testing document selection methods .................................................. 92
5.3 Average RankNet gradients by document rank . . . . . . . . . . . 101
5.4 Comparison between methods that do not use top-ranked docu-
ments on MSLR-WEB30K collection . . . . . . . . . . . . . . . . . 103
5.5 Comparison between methods that do not use top-ranked docu-
ments on Yahoo! LTR collection . . . . . . . . . . . . . . . . . . . 104
5.6 Comparison between the methods depth pooling and statAP on
MSLR-WEB30K collection . . . . . . . . . . . . . . . . . . . . . . . 105
5.7 Comparison between the methods depth pooling and statAP on
Yahoo! LTR collection . . . . . . . . . . . . . . . . . . . . . . . . . 106
5.8 Comparison between methods that use top-ranked and random doc-
uments on the MSLR-WEB30K collection . . . . . . . . . . . . . . 111
5.9 Comparison between methods that use top-ranked and random doc-
uments on the Yahoo! collection . . . . . . . . . . . . . . . . . . . 112
5.10 Comparison between methods that use top-ranked documents on
the MSLR-WEB30K collection . . . . . . . . . . . . . . . . . . . . . 113
5.11 Comparison between methods that use top-ranked documents on
the Yahoo! collection . . . . . . . . . . . . . . . . . . . . . . . . . 114
5.12 Effect of selecting top documents in the dataset using either BM25
feature score or the relevance score of a regression model for the
MSLR-WEB30K collection . . . . . . . . . . . . . . . . . . . . . . . 115
5.13 Comparison of multiple data selection methods on the MSLR-WEB30K
collection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 116
5.14 Comparison of multiple data selection methods on the Yahoo! LTR
collection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117
List of Tables

1.1 Some characteristics of MSLR-WEB30K and Yahoo! datasets . . . 15

2.1 Classification of learning algorithms across the dimensions of loss
function and concept space . . . . . . . . . . . . . . . . . . . . . . . . . 24

4.1 Correlation between the distance between the distributions of rele-
vant and non-relevant documents and NDCG@10 . . . . . . . . . . . 76

4.2 Interaction between feature variability and relevance categories . . 78

5.1 Some characteristics of the learning algorithms . . . . . . . . . . . 93

5.2 Parameters of the data selection methods . . . . . . . . . . . . . . 96
Acknowledgments

First and foremost, I would like to thank Vessela Hristova for her moral support and for her help in editing the text. I would like to thank my advisor Javed Aslam for his mentoring and financial support. Many thanks to Professors Ravi Sundaram, Mirek Riedewald and Ben Carterette for serving on my committee. I would also like to thank my graduate school coordinator Bryan Lackaye for handling smoothly all the paperwork challenges during the final stages of my thesis. I would like to thank all my friends from the Information Retrieval lab for the good time at Northeastern. Special thanks to Pavel Metrikov for letting me use his RankBoost implementation and to Evangelos Kanoulas for his leadership in writing papers.
1 Introduction

Significant research and development efforts are being devoted to the improvement of web search engines due to their increasing importance in our society. Traditionally search engines relied on relatively simple text-matching models. The complexity of web search, however, necessitated the development of more sophisticated approaches to the problem of ranking documents. Current approaches are based on a combination of 400–1000 ranking signals (features). Some of the signals are text-matching features such as the well-known BM25 text matching formula applied to the document text. Other signals such as PageRank are related to web page importance. A third group, clicks, is collected through the interaction of users with the search engines. How to most meaningfully combine those signals to produce a single score per document still remains a question open for further research.

One general method for combining signals into a ranking function is based on Machine Learning. In the Information Retrieval literature, this approach is known as learning to rank. The machine learning algorithms take as input a set of data-points known as the training set. Each data point in the training set is an N-dimensional feature vector. The task of a learning algorithm is to find rules for combining the feature values into a scoring function predicting the relevance of a document. The derived rule constitutes a machine learned model that is evaluated on an independent test set with documents that have the same kind of features. A number of variables play an important role in the process of obtaining a ranking function through machine learning: the training dataset, the learning algorithm, the test dataset and the methods to measure the quality of the ranking function (evaluation). Many of the components of learning to rank such as the learning algorithm, test set construction and the evaluation methodology have been well-studied. However, questions concerning the role of the training dataset in the
training process have received little attention from the research community so far. Nevertheless, the training data plays an important role in machine learning as the learners can only learn relationships that are present as examples in the training set. It is known from the field of machine learning that the machine learning methods become more effective as the size of the training data increases. In learning to rank, however, one bottleneck in obtaining training data is that the data actually needs to be annotated by humans. Thus, the task of obtaining a training set can be formulated as the selection of a given number of documents for labeling that would constitute the training collection on which learning algorithms achieve the best performance among training collections of the same size. The quantity of training data is related to the efficiency aspect of training collection construction, while the quality of the gathered data is related to the effectiveness of the training collection. In this thesis I evaluate and develop methods for efficient and effective construction of training collections for learning to rank. My thesis answers questions such as what properties make a training dataset good for learning to rank and how to construct training datasets that are better than existing baselines. I use the theory of Optimal Design of Experiments (ODE) to develop theoretical propositions about the properties of good training collection. At the same time, I rely on a sound empirical approach to verify the properties of good training collection and evaluate proposed collection construction methods. I have found that the best performing collections have a combination of relevant and non-similar (diverse) documents. The optimal combination of those two properties depends on (1) on the collection size and (2) on the learning algorithm. When the collection grows in size, the percentage of relevant documents should also grow. Algorithms using non-linear feature combinations achieve better performance when documents in the collection are less similar. The definition of non-similar documents is algorithm dependent, and according to ODE it is related to the definition of a kernel function between two data points. The kernel function specifies how two data points influence each other and intuitively high values of the kernel function can be equated with higher similarity between the data points.
One definition of similar documents that has worked well in my thesis is the dot product between the document feature vectors.

1.1 Background Information

1.1.1 Search engine architecture

The architecture of web search engines consists of several main components: document crawlers, documents indexers, document rankers and a user interface. The crawler is responsible for choosing what web documents to retrieve and store in the search engine and for actually fetching those documents from the web. The web documents are stored in a database in an optimized format known as the index. When the user issues a query, the query is parsed and in many cases augmented with additional information. The preprocessed query is scored against a large number of documents which are considered reasonable candidates. One simple strategy to obtain an initial set of candidates is to take the documents which have at least one word in common with the query. A more complicated strategy involves pre-scoring with a simple and more efficient ranking function to obtain the initial candidate set. Each document in the initial set of candidates is represented as feature vector. The features can be classified in a number of groups such as query features depending only on the query, document matching features depending on the document text and static document features such as document popularity. The final score for each document is produced by transforming and aggregating the document features by means of a scoring function. Such a function could be hand-tuned or derived through machine learning. After every candidate document is given a score, the list of candidates is sorted by score and the documents with the top scores are presented as best matches to the user.

1.1.2 Information Retrieval

The academic discipline that studies techniques for implementing search engines is Information Retrieval (IR). IR is a broad field concerned with automatic
Although there are potentially infinite number of queries and documents that can be selected for training, the actual training and test sets are finite. The documents chosen in the training and test sets need to be labeled with their degree of relevance.
methods for organizing mainly textual information. Apart from textual search, other important themes of IR research relate to clustering, summarization of textual documents, and question answering.

In the study of search engines, there are at least three very important components: the efficacy of the methods, their efficient implementation, and presentation of results to the user. The efficacy of search implementation is concerned with methods, models and frameworks that will bring relevant information to the user. The efficiency aspect deals with index organization, data compression and scalability using distributed implementations. The interaction with the user can be organized in multiple ways. The questions to the search engines can be posed through keywords, natural language, or using a predefined taxonomy. The presentation of results can be achieved through a flat list of results, hierarchy, or clusters and include features such as spelling correction, related queries and suggestions for alternative query formulations.

1.2 Learning to Rank

Machine learning applied to ranking tasks in Information Retrieval is known as learning to rank. Learning to rank has been used by some big search engines companies such as Bing, Yahoo! and Yandex. Google currently uses experts to hand-tune their algorithm, but they have also trained machine learning models competitive to the hand-tuned formula.¹

The academic community has conducted extensive research on the learning to rank problem. Mostly attention has been paid to deriving sophisticated learning algorithms. A large number of learning algorithms are currently available for learning to rank including methods based on Regression, Support Vector Machine, Boosting and tree induction. The current state-of-the-art among machine learned ranking methods is LambdaMart. LambdaMart is based on the optimization of a specific information retrieval measure through boosting. Bing currently uses a

¹http://www.webcitation.org/5sq8irWNM
method based on LambdaMart. Other methods based on regression trees have been used in AltaVista around 2002 and are currently also used in Yandex.

Learning to rank involves the following parts:

- creation of training collection
- creation of test collection
- choice of learning model
- choice of evaluation measure
- application of the learning model on the training collection
- validation of obtained model on the test collection

The learning to rank setup is illustrated in Figure 1.1. I describe the component of the learning to rank process below.

### 1.2.1 Test collections

Historically test collections were developed to evaluate ranking methods that do not rely on learning. However, today the methods for test collection construction are instrumental to the learning to rank process as well.

Test collections have played an important role in the advancement of Information Retrieval since the introduction of the Cranfield paradigm in the 1950's. The paradigm establishes a methodology for comparison of different retrieval systems using a common reference collections. Test collections help to establish the reliability and reproducibility of experimental results.

The test collection comprises of a set of documents and queries. Each query has a list of documents annotated with relevance labels. The relevance labels were produced by judges examining the document text against the query. For each

---


3 [http://jopedersen.com/Presentations/The_MLR_Story.pdf](http://jopedersen.com/Presentations/The_MLR_Story.pdf)

query there is a much larger number of unjudged documents than judged ones. Typically, the unjudged documents are considered non-relevant.

The early collections were quite small both in terms of number of queries used for evaluation and in terms of number of documents in the collection. When the collection contains smaller number of documents, simpler models, for example boolean search, may return adequate results, since very few documents will contain the query words. Thus, a method performing well on a small collection will not necessarily perform well on a larger collection. So, large collections of documents are required to evaluate powerful ranking methods. The number of queries in the test collection influences the statistical validity of the results. Larger test collections became necessary as the field on Information Retrieval progressed and the Text Retrieval Conference (TREC) was created to coordinate the effort. TREC was created in 1992 by National Institute of Standards and Technology and U.S. Department of Defense to provide the methodology and infrastructure necessary for the development and large-scale evaluation of Information Retrieval technology.

TREC has established a process through which they release new test collections. The process involves participants mainly from academic institutions but sometimes also from commercial corporations. TREC is responsible for the set of queries and a collection of unjudged documents provided to the participants. Each participant develops their own search algorithm and applies it to the provided data. At this stage the documents do not have any relevance annotations. After each participants submits their results, TREC uses assessors to judge a large number of the documents retrieved by TREC participants. Traditionally, TREC used the depth pooling method to judge some portion of the retrieved documents. Under the depth pooling strategy, the top 50 documents from each participant’s search engine are put into a common pool and the union of documents from all participants are judged for relevance. It can be seen that the quality of the judged documents and of the produced test collection depends on the number, quality and diversity of search engines that have participated in TREC.
1.2.2 Relevance judgments

Relevance judgments are necessary both for training and test collections. The definition of relevance most commonly used in Information Retrieval is operational: a document is relevant if the user’s question is answered by the information that they find in the presented documents. During the development, tuning and evaluation of search engines real users are replaced by document assessors (judges). The judges are given instructions which state what documents to consider relevant. The typical setup is that a judge will be presented with a query and a document and he/she will decide on the degree to which the document will satisfy the user’s request. This setup poses some challenges. Will the judge be able to understand the user’s intent from the few query words? Will the judge be an expert in the subject matter to judge properly the search engine response? There are a number of studies which explore the question of the quality and the effect of relevance judgments. For example, it is known that judges tend to disagree on what is relevant and even the same judge may give different relevance labels at different times.

Originally relevance judgments used to be binary: relevant and non-relevant. Today, in learning to rank tasks the judgments have five grades:

\{bad, fair, good, excellent, perfect\}

The grades are represented numerically from 0 to 4 with 4 representing the perfectly relevant documents.

In the past, relevance labels were used only for search engine evaluation. Currently, relevance labels are used both for development of models and during evaluation of search engine quality. The quality of the labels has become extremely important.

Various efforts have been made to reduce the effect of noise in the judgment process. One way to address the problem is in the design of the collections through better allocation of judges and queries. Another approach is to change the unit of judging by considering comparative judgments based on pairs of documents rather
than document-level judgments. The effect of label noise could also be accounted for when relevance judgments are used in learning to rank.

### 1.2.3 Evaluation

The methodology of evaluation in Information Retrieval is concerned with evaluation measures. One of the most popular evaluation measures in Information Retrieval is average precision. To define average precision we first need to define precision, recall and precision at cut-off \( k \). Precision and recall are defined for sets of documents, not ranked lists. Given a set of retrieved documents, precision is equal to the number of relevant documents in the set:

\[
\text{precision} = \text{prob}(\text{relevant}|\text{retrieved}) = \frac{\text{relevant} \cap \text{retrieved}}{\text{retrieved}} \quad (1.1)
\]

Recall is equal to the number of relevant documents that were retrieved.

\[
\text{recall} = \text{prob}(\text{retrieved}|\text{relevant}) = \frac{\text{relevant} \cap \text{retrieved}}{\text{relevant}} \quad (1.2)
\]

Precision at cut-off \( k \) is defined for a ranked list of documents. It is the precision for the set of documents at ranks 1, 2, \ldots, \( k \). Average precision (AP) is the average of the precision at cut-offs where relevant documents occur. If in a ranked list relevant documents occur at ranks \( r_1, r_2, \ldots, r_n \) and \( R \) is the total number of relevant documents, then average precision is equal to

\[
\text{average precision} = \frac{1}{R} \sum_i \frac{i}{r_i} \quad (1.3)
\]

Mean average precision (MAP) is AP averaged over all queries. MAP and AP used to be the most popular measures for Information Retrieval evaluation but they were surpassed in popularity by the NDCG measure due to the importance of NDCG in web search. While average precision is useful for ranking problems with binary relevance, NDCG is typically used for ranking problems with multigraded relevance.
NDCG stands for normalized discounted cumulative gain. To define NDCG, one needs to define DCG first and then normalize it. Assume a ranked list of documents with $r_i$ being the relevance label of document $i$ at rank $\pi_i$. Then DCG at rank $k$ is equal to:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{r_i} - 1}{\log_2(\pi_i + 1)}$$  \hspace{1cm} (1.4)

NDCG is obtained by dividing DCG by the maximum possible NDCG.

$$NDCG_k = \frac{DCG_k}{\text{Ideal } DCG}$$  \hspace{1cm} (1.5)

NDCG consists of two series of weights: one due to the labels and the other one due to the ranks. Highly relevant documents at high ranks are rewarded with huge gains.

Traditional evaluation approaches rely on depth pooling: extensive judgment of all documents up to rank 50 across a large number of search engines. The effort required to produce judgments can be reduced if smarter strategies are used. Inferred Average Precision (infAP) defines one such strategy: infAP is an estimate of AP with confidence intervals. Other important methods for efficient and effective evaluation in test collections that do not require judgments of all documents are statAP [1] and MTC [2]. Both statAP and MTC evaluation methods were found to work well in a large scale “Million Query” study supported by TREC [3].

1.2.4 Training collections

Learning to rank training collections consist of query-document pairs. For each query in the dataset there might be 10 to 1000 labeled documents. Each document is represented as a feature vector with a large number of features. For example, the Microsoft MSLR-WEB30K dataset\(^5\) has 136 features while the dataset from the Yahoo! Learning to rank challenge has 700 features. The relevance labels in

\(^5\)http://research.microsoft.com/en-us/projects/mslr/
MSLR-WEB30K use five grades.

The construction of training collections has two stages (1) the choice of queries and unlabeled documents and (2) the labeling of documents. The procedure of labeling the documents in terms of their relevance is time-consuming and labor-intensive. Efficient training collections minimize the amount of documents that need to be labeled to obtain learning algorithms of given accuracy. The effectiveness of training collection is related to the choice of the queries and documents. Effective training collection should use documents and queries from which the learning algorithm can learn the most (the “most informative documents”).

The construction of training collection begins with the choice of queries and documents from an extremely large pool of queries and unlabeled documents. The universe of queries consists of all possible queries that one can formulate. In the old TREC collections the queries were created as part of the collection design. Today millions of queries are available for selection from the search engine query log and a number of them are chosen typically based on query frequency to participate in the training set. The possible unlabeled documents are all documents from the web and any of those could potentially be useful for the machine learning procedures. For most queries there are a small number of relevant documents and an extremely large number of non-relevant documents. It does not make sense to select documents which are completely unrelated to the queries since they would be non-relevant. In practice, the document selection is informed by some kind of scoring of documents based on the query.

The second stage of training collection construction is the labeling of every document in the training set with its degree of relevance vis-à-vis a given query. The documents that are a good match to their respective query are labeled as relevant. Other documents might provide a less useful answer to the query and some, like spam, will have a negative effect on the user’s perception of the quality of the search engine. The documents have a range of usefulness to the users, which can only be determined by manual inspection of the pair of document and its query.
Chapter 1. Introduction

A number of collections for learning to rank are currently available to researchers. The older training collection by TREC such as TREC 6, TREC 7 and TREC 8 were quite frequently used in the past to evaluate newly proposed retrieval methods. The documents in those collections are selected by scoring all potential candidate documents for a query by a large number of search engines and then using depth-pooling. Unfortunately, although those collections contain many documents, those are distributed across a small number of queries (e.g. 50). This property of the collections makes it unsuitable for machine learning.

The earliest attempt to create a principled collection for learning to rank tasks was LETOR1 [4]. LETOR1 was formed by pre-judged documents from TREC and OSHUMED collections. The documents were selected in two stages. In the first stage documents were ranked by BM25 score and the top 1000 documents were selected. Second, relevant documents that were not retrieved by the BM25 text-based model were added. The rationale for this method can be justified: relevant documents are more important than non-relevant, so to increase the number of relevant documents all available relevant documents were included. However, a problem occurred in the selection of non-relevant documents because the only non-relevant documents that were included were the ones with sufficiently high BM25 score. Since many relevant documents did not have a high BM25 score, but many of the included non-relevant documents did have a high BM25 score, relevance was inversely correlated with BM25. As pointed out in [5] this finding is unexpected since BM25 is one of the most important features in ranking. Therefore, LETOR1 is considered an unsuccessful attempt to create a collection for learning to rank.


Apart from LETOR, the Yahoo! learning to rank challenge [7] released a collection suitable for large scale evaluation of learning to rank algorithms. This collection was originally released to participants in the Yahoo! learning to rank challenge. One of the goals of the challenge was the release of a dataset similar to the datasets used by large search companies. The training dataset consisted of around 20,000 queries and almost a million documents. The dataset creation
followed a two-stage process. In the first pass, the queries were sampled from the query log according to their frequency. Frequent queries appear in the set multiple times. In the second stage, documents for each query are selected. The Yahoo! dataset uses the traditional pooling strategy from TREC: a number of diverse search engines are used to rank the documents for each query and then the union of top $k$ documents is selected. In the Yahoo! collection $k$ was set to 5. Such a collection construction method would imply that easy queries have a small number of documents selected while difficult queries will have a large number of documents. The average number of queries was 24 but there were queries with more than 100 documents.

The latest LETOR collection, named MSLR-WEB30K, was based on an existing collection used by Microsoft Bing search engine that has been retracted from use. The features in this collection were features known from published research rather than proprietary features.

The original LETOR dataset was based on TREC data and lacked a number of features commonly used by web search engines. To address this short-coming the MSLR-WEB30K collection is based on data used by Microsoft Bing search engine.

### 1.3 Related Work

A number of studies are available that evaluate methods for training collection construction. The study by Aslam et al. [8] showed the effect of different methods developed for test collections when utilized as training collections construction methods. The Aslam et al. study used the following test collection construction methods: depth-pooling, statAP, infAP and MTC. Aslam et al. identified the following important factors for LTR collections: a) the precision and recall of the constructed datasets, and b) the textual dissimilarity between included documents from different relevance categories. When the number of either relevant or non-relevant documents was too small, the performance of learning algorithms degraded. The presence of many similar documents also caused drop in algorithm
performance.

Tian et al. [9] proposed a criterion based on minimization of the similarity between pairs of documents in the context of SVM. Their criterion leads to the selection of pairs of documents with feature vectors far apart.

Yilmaz [10] suggested that when the same amount of judged documents is allocated over more queries, the performance of a learning to rank algorithm is improved.

Training collection play an important role in commercial search engines, but little is known about how the commercial search engines have created the datasets they use for tuning or training a ranker. It has been suggested that Yahoo! used importance sampling for their collections: positive examples (relevant documents) are emphasized while non-relevant ones can be sampled.6

1.4 Experimental Data

In my experiments I use the MSLR-WEB30K dataset available from Microsoft Research.7 This dataset consists of over 30000 queries sampled from a retired log from Microsoft Bing Search Engine. The query labels are on a scale from 0 to 4, with 0 denoting a non-relevant document and 4 denoting a highly relevant document. Some useful statistics of the used dataset appear in Table 1.1. The dataset has 136 features. The features fall into three broad categories: textual features, document quality features, and clicks. The textual features consist of a number of well-known textual retrieval methods such as raw counts, BM25 and language models computed over a number of different textual streams such as url, links, title, and body text. The full list of features is available on the collection website8.

---

6http://jopedersen.com/Presentations/The_MLR_Story.pdf
Chapter 1. Introduction

1.5 Research Questions

The main research question that this study addresses is how to efficiently create effective training collections for learning to rank. The efficiency aspect is related to the amount of training data used while the effectiveness aspect implies that the machine learning algorithms should perform well on the given training sets. I treat the problem of collection construction as one in which a set of queries and documents for those queries need to be selected from a large but fixed finite pool. Web-scale search engines contain billions of queries in their query logs. Therefore the query log can be used as a source of queries. The top 200 to 500 documents as ranked by the currently deployed ranking function can be used as a source for document selection. Once the selection process is complete, the documents are labeled and the collection is evaluated for quality using multiple learning algorithms on a fixed test set. A number of interesting questions arise in this setup.

<table>
<thead>
<tr>
<th>Documents per query</th>
<th>MSLR-WEB30K</th>
<th>Yahoo! LTR collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Max</td>
<td>1251</td>
<td>139</td>
</tr>
<tr>
<td>Avg</td>
<td>120</td>
<td>24</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Relevance categories</th>
<th>MSLR-WEB30K</th>
<th>Yahoo! LTR collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat. 4</td>
<td>18231 (0.8 per)</td>
<td>5313 (1.86 per)</td>
</tr>
<tr>
<td>Cat. 3</td>
<td>41363 (1.82 per)</td>
<td>21542 (7.57 per)</td>
</tr>
<tr>
<td>Cat. 2</td>
<td>303502 (13.3 per)</td>
<td>80921 (28.46 per)</td>
</tr>
<tr>
<td>Cat. 1</td>
<td>737177 (32.5 per)</td>
<td>101811 (35.81 per)</td>
</tr>
<tr>
<td>Cat. 0</td>
<td>1170023 (51.5 per)</td>
<td>74687 (26.27 per)</td>
</tr>
</tbody>
</table>

Table 1.1: Some characteristics of MSLR-WEB30K and Yahoo! datasets

First, what effect does the training collection have on the obtained ranking function? In Chapter 5 I show that reasonable data selection strategies perform differently. Therefore the data selection method has a non-trivial effect over the learned models.

Another important question concerns the properties of the better performing
training collections. In Chapter 4 I systematically study the effect of a few important properties on the quality of the training collection. The properties that I study are related to the representation of relevant and non-relevant documents, to the ratio between the number of documents and the number of queries and to the effect of the feature values. My approach to those studies is to generate a large number of training collections each having different values of the studied property. One general conclusion is that the value of the training collection depends not on the properties of a particular document, but on the configuration between documents present in the collection. A trivial illustration would be two collections: one containing mostly relevant documents and the other containing mostly non-relevant documents. By combining both collections we are likely to obtain a much better training collections than the original ones.

A third question that I consider is whether the properties of the good training collections can be theoretically explained. In other words, can an optimal training construction algorithm be derived from a theoretical perspective. This question leads me to the use of the theory of Optimal Design of Experiments from Statistics (ODE). The analysis of the theory shows how to construct theoretically optimal training sets for a given learning algorithm. Depending on the algorithm, the constructed training sets are different. In Chapter 3 I explore the ODE theory and use it to explain that the data selection depends mainly on two features of the learning algorithm: the optimized target function (squared error, pair-wise error, etc.) and the representation of the feature space (linear vs. non-linear feature combinations). Based on the insights from Chapter 3 I investigate the use of some ODE-informed data selection methods and compare them to strong baselines in Chapter 5.

Through a combination of my practically motivated study of the properties of the training collections and the theory of ODE, I am able to provide an answer to a puzzle that has appeared in the literature of training collection construction. In Aslam et al. [8], it was shown that training collection in which the set of relevant documents was better separated from the set of non-relevant docu-
ments performed better. This finding has been considered counter-intuitive from the point of view of active learning. Some active learning data selection procedures would iteratively select training data points from both classes of a binary classification problem such that the selected points are closer to the estimated decision boundary. Therefore, when using active learning procedures, the selected relevant and non-relevant documents would tend to be closer rather than apart. One explanation of why larger distance between the distribution of the relevant and non-relevant documents correlates to better training collections is that when the distributions are better separated the learning algorithm is less influenced by noise and is able to find more stable models.

Revisiting the original question of how to create efficient and effective training collections, I explore a number of collection construction algorithms in Chapter 4 and Chapter 5. I differentiate between methods that have access to some estimate of the relevance of the documents and methods that do not. In practical situations estimates of the relevance labels would be available from previous versions of the ranking algorithm. Although the labels suggested by an automatic methods are not correct, they would offer some indication of what documents are relevant. I conclude that relevance estimates are critical for creating training collections.

In my study I evaluated methods based on ODE as well as some ad-hoc methods. My conclusion is that when the dataset contains both (1) relevant documents and (2) non-similar documents in terms of features, the learning algorithms perform the best. Algorithms non-linear in the feature space require documents to have less similar features. Regarding the relevance aspect, I found that the “optimal” ratio of “classified-as-relevant” documents to all documents in the collection, increases as the collection size increases. In other words, relevant documents are more important for larger collections rather than for smaller collections.
2 Information Retrieval Methods and Learning to Rank Algorithms

In this chapter I present a number of well-known algorithms for ranking documents by their relevance. Traditionally, those algorithms were not based on machine learning, but as the stakes and the scale of the ranking problem in web search increased, the machine learning approach took over.

The traditional ranking algorithms are still of interest since the ranking scores produced by a simple ranking algorithm can be used as the values of a single feature in the machine learning approach.

The presentation of machine learning approaches serves two purposes here. First, the learning to rank paradigm in Information Retrieval cannot be understood without firm grounding in the theory of learning algorithms. Secondly, it turns out that the problem of creating effective datasets for learning to rank cannot be understood well unless the algorithms that operate on the datasets have been understood.

As I demonstrate in Chapter 3, data selection depends on the properties of the learning algorithm. In this chapter, I describe concisely a number of important algorithms. I have chosen the presented algorithms based on two criteria: (1) the learning algorithm popularity in the learning to rank literature and (2) the need to show diverse algorithms across a number of dimensions. In Table 2.1 I group the algorithms by two of the most important dimensions in the learning problem: the (1) loss function and (2) concept/hypothesis space. It turns out, as I show in Chapter 3, that those dimensions have fundamental influence on the “optimal dataset”.

2.1 Information Retrieval Models

The retrieval models attempt to capture the concept of relevance. The earliest models were the boolean model [11] and vector space model [12]. Subsequent better performing and theoretically justified models were based on probability theory.

2.1.1 Early models

In the boolean model, documents cannot be ranked because the model returns a set of documents that matches a boolean predicate defined by the query. The vector space model represents both documents and queries in a high-dimensional space. The number of dimensions equals to the number of words in the training corpus. Both queries and documents are represented as vectors in the same vector space. The components of the vectors are related to the number of times a word occurs in the document and its importance. The vector space model is a “bag of words” model because it discards the positions of the words in the document. The similarity between a query and a document can be defined as the cosine of the angle between the vectors representing the query and the document. Documents whose vectors are closest to the query in terms of angle, are considered the most relevant.

2.1.2 Text-matching models based on probability

Some of the most important text-matching models are based on probability theory. Among those model are:

- models based on the probability ranking principle;
- models based on language modeling;

Although attacking the retrieval problem with different theoretical tools the conclusions of the models from different paradigms are qualitatively the same: (1)
terms are of different quality; (2) the number of times a word occurs in a document matters and (3) the retrieval function should control for document length. Each word from the query contributes differently to the relevance of a document, but rare words are more informative about the intention of the searcher and contribute more. When a query word appears more frequently in a document, it contributes more to the document being relevant although each additional occurrence contributes less. Document length normalization is crucial to penalize long documents which have an unfair advantage in matching the query.

**Probability Ranking Principle**

One of the best-performing models based on the probability ranking principle (PRP) [13] is the so called Okapi BM25 model. According to PRP, the optimal ranking of documents is achieved when documents are ranked according to their probability of relevance. The probability of relevance is used as a way to model the usefulness of the document to the user. A particular instantiation of Okapi BM25 after giving concrete values to certain parameters reads:

\[
\text{okapi-bm25} = \sum_{w \in \text{query}} \text{idf}(w) \times \text{tf-weight}(w) \tag{2.1}
\]

\[
\text{idf}(w) = \log \frac{N - \text{df}(w) + 0.5}{\text{df}(w) + 0.5} \tag{2.2}
\]

\[
\text{tf-weight}(w) = \frac{2 \times \text{tf}(w)}{\text{tf}(w) + (0.25 + 0.75 \times \frac{\text{doc-len}}{\text{avg-doc-len}})} \tag{2.3}
\]

The Okapi BM25 formula shows that each query word contributes additively to the total score. The contribution of any word depends on two components: \(\text{idf}(w)\) and \(\text{tf-weight}(w)\). Idf stands for inverse document frequency. It has a high value for rare words and quantifies the intuition that rare words are more important in search.

The score of a word also depends on document frequency, \(\text{df}(w)\), which is defined as the number of times the word \(w\) occurs in the corpus. The tf-weight component is based on term frequency \(\text{tf}\), which is defined as the number of times the word occurs in the document. The higher the term frequency, the larger the
contribution of this term. Examination of \( tf\text{-}weight(w) \) shows that each successive occurrence of a term in the document brings less value than the previous occurrence. The Formula 2.3 accounts carefully for documents of different lengths. The idea behind the document length normalization is that documents whose length is below a threshold should have their scores increased relative to a baseline model, while documents with lengths above the threshold should have their scores increased.

To this date, Okapi BM25 remains one of the most important contributions in Information Retrieval and is used as a feature in many ranking functions obtained through machine learning. Important extensions of BM25 exist which can make use of multiple textual streams and positional information.

### Language Models

Language modeling provides a different perspective of modeling the utility of a document for retrieval. The idea is that the query can be modeled as the result of a process in which a user reads a document and summarizes it through a query. There are various ways to measure the probability \( p(q|d) \) of the query given the document, for example Jelinek-Mercer smoothing:

\[
score = \sum_{w \in \text{query}} \log P(w|d)
\]  
\[
P(w|d) = \lambda \frac{tf_{doclen}}{doclen} + (1 - \lambda) \frac{ctf}{collectionsize}
\]  

The score of a document is equal to the negative of the log of the probability that the query is generated by this document. Under Jelinek-Mercer smoothing, the probability of a word to be generated from the document is an interpolation between the unsmoothed probability of the word in a document and the probability of a word in the collection. This interpolation is critical in order to obtain weights for each word, which similar to the idf component in BM25 [14], depend on the rareness of the word in the collection.
2.2 Learning Algorithms

Learning to rank algorithms have their roots in the discipline of Machine Learning. Machine learning algorithms have been studied in both theoretical and applied settings by fields such as Statistics, Pattern Recognition, Machine Learning and Data Mining. The goal of the learning algorithms is to build a predictive model based on a set of labeled data points. As an example consider the task of predicting whether a person will fall seriously ill in the upcoming year. It is possible to build a predictive model based on attributes including the prior health history of the person, genetic factors, work conditions, age, pollution, and so on. In order to build the predictive model, a large and representative dataset needs to be created which relates the predictive attributes to the predicted variable. Once the model is built, its usefulness should be assessed of an unseen test set. It is very important that the learned model generalizes well. This means that when the model is deployed in practice it should exhibit performance similar to the performance observed in the lab. It should be noted that theoretical performance guarantees apply as long as the real data comes from the same distribution as the data used in training and evaluation.

In machine learning, some of the most common learning tasks are classification, regression and ranking. While the field of Statistics has mostly focused on regression, the machine learning community has been mainly interested in the classification problem. The ranking problem emerged most recently due to the needs of disciplines such as Information Retrieval.

Some important questions regarding a machine learning model are (1) the objective on which the model is evaluated; (2) the objective on which the model is trained, (3) the concept/hypothesis space (e.g. linear, non-linear, tree-based, etc.) (4) feature selection and feature transformations and (5) techniques such as bagging that reduce the variance of the predictions and improve the generalization ability of the model (decrease overfitting).

Some popular evaluation objectives for machine learning models are the error rate, negative log-likelihood, the mean squared error, absolute error, area under
the ROC curve and area under the precision-recall curve. The objectives optimized during training could be the same as the evaluation objectives, but not always. For example, it is common to train a classifier using negative log-likelihood but evaluate it using error rate. On the one hand, it could be argued that better performance could be obtained by using the same measure for learning and evaluation. On the other hand, there are issues of non-differentiability (non-smoothness) of measures and the variance they cause in a model. For example, training on the error rate directly may result in more unstable models that training using likelihood criteria.

The learned model depends on the training dataset. The training dataset is a sample (either random or biased) from the true population which is being modeled. The predicted variable that is learned may also be measured with noise. Those factors cause instability in the machine learning algorithms. There is a trade-off between the accuracy of the model and its stability. This issue is known as the bias-variance trade-off in the literature. Consider a prediction made at some test example. The prediction will be different every time a different model is used. If on average the prediction equals the true label of the example, then the bias is zero. So, the bias refers to the extent to which predictions tend to be systematically off. In ridge regression, for example, the predictions are biased towards the mean of the predicted variable. In that case, the variance of the predictions is reduced and the overall accuracy maybe increased. Techniques that decrease both bias and variance do exist in the machine learning literature. Bagging is one technique for variance reduction that works averaging the predictions over multiple and diverse learning models. Bias can be reduced by the use of more flexible learning models.

Some well-known methods that I use in my dissertation are linear regression, logistic regression, neural networks, support vector machine, ada-boost and boosted decision trees.


<table>
<thead>
<tr>
<th>Loss</th>
<th>Linear</th>
<th>Non-linear</th>
<th>Using Feature Splits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Squared Loss</td>
<td>Regression</td>
<td>Gaussian Process</td>
<td>Regression Tree</td>
</tr>
<tr>
<td>Classification</td>
<td>SVM</td>
<td>SVM with Gaussian Kernel</td>
<td>Ada Boost</td>
</tr>
<tr>
<td>Loss</td>
<td>Logistic Regression</td>
<td>Kernel Logistic Regression</td>
<td>Decision Tree</td>
</tr>
<tr>
<td>Pairwise</td>
<td>Ranking SVM with linear kernel</td>
<td>Ranking SVM with Gaussian Kernel</td>
<td>Lambda Mart</td>
</tr>
<tr>
<td>Loss</td>
<td>RankNet with 1-layer neural network</td>
<td>Lambda Rank with 2-layer neural network</td>
<td>Rank Boost</td>
</tr>
</tbody>
</table>

Table 2.1: Classification of learning algorithms across the dimensions of loss function and concept space.

### 2.2.1 Linear regression

Under linear regression the predicted variable $y$ is expressed as a weighted linear combination of the features $x^T$ and additional noise:

$$y = x^T \beta + \epsilon$$  \hspace{1cm} (2.6)

The weights $\beta$ represent the model to be learned. There is a closed-form solution of the optimal beta:

$$\beta = (X_{train}^T X_{train})^{-1} X_{train}^T y_{train}$$  \hspace{1cm} (2.7)

where the matrix $X_{train}$ is an $n \times m$ matrix for $n$ training examples each with $m$ features. The rows in this matrix are the feature vectors. In practice, the above unbiased estimate of the true $\beta$ does not perform well on predictions. Under ridge regression the estimate is shrunk towards 0 in the following way:

$$\beta = (X_{train}^T X_{train} + \lambda I)^{-1} X_{train}^T y_{train}$$  \hspace{1cm} (2.8)
The effect on the predictions is that they are shrunk towards the mean of the label.

Regression is the work-horse of other more sophisticated algorithms like logistic regression and boosted decision trees and has been linked to the theory of Gaussian Processes [15] and Support Vector Machines.

### 2.2.2 Logistic regression

Logistic Regression is a classical tool for solving binary classification problems. The assumption made in logistic regression is that the log of the odds ratio of the response variable can be expressed as a linear combination of features:

\[
\log \left( \frac{y}{1 - y} \right) = \beta^T x
\]  \hspace{1cm} (2.9)

An alternative way to express the logistic regression model is to express the probability of success through the sigmoid function:

\[
y = \text{prob. of success} = \frac{1}{1 + e^{-\beta^T x}}
\]  \hspace{1cm} (2.10)

Estimation of the parameters is based on maximum likelihood. Taking derivatives of the log-likelihood and applying Newton’s method for iterative optimization leads to an iteratively applied least squares algorithm.

One issue with training logistic regression is that when the iterative solution is applied, the probabilities of class membership \( p \) and \( 1 - p \) can be driven to their extreme values of 1 or 0. This implies a division by a number close to zero in the estimation procedure. Another way to interpret this situation is that the model becomes too confident in certain predictions, and this is, as a rule of thumb, a sign of overfitting the data. One solution in this case is to apply a form of regularization.

The usefulness of logistic regression is not so much in creating accurate prediction models but in the interpretability of the model. For example, under the assumptions of logistic regression, it is possible to calculate how much a change
of one unit in a predictor will change the probability of the outcome. Closed form formulas for the variance in the estimated coefficients and predicted outcomes are also available.

### 2.2.3 Neural networks

Neural networks are layered models for learning non-linear transformations. A single-layer neural network for regression could be described by the equation:

$$ y = \sum_i \sigma(\beta_0 + \beta_i^T x) $$  \hspace{1cm} (2.11)

with $\sigma$ typically being the sigmoid or the tanh function. The sigmoid is the function $\sigma(v) = \frac{1}{1+e^{-v}}$ also used as a link function in logistic regression. A two layer network is defined by:

$$ z_i = \sigma(\beta_{0i} + \beta_i^T x), i = 1 \ldots m $$  \hspace{1cm} (2.12)

$$ y = \sum_{j=0}^k \sigma(\alpha_{0j} + \alpha_j^T [z_1, z_2, \ldots, z_m]^T) $$  \hspace{1cm} (2.13)

In a two-layer network, the input $x$ is first transformed to $m$ intermediate outputs $z_i$ as in Equation 2.12. Then $k$ differently weighted combinations of $z_i$ are summed to obtain the final score of the network $y$ as in Equation 2.13. The most common way of training the network is through gradient descent. The gradient is computed by a method called back-propagation. The training procedure starts with an initial guess of the parameters $\beta$ and $\alpha$ to be learned. At every iteration the current solution is moved a bit in the direction of the negated gradient. The computation of the gradient starts at the error function which for regression is usually squared error loss and for classification is the cross-entropy loss. The gradient is propagated from outputs to inputs starting at the output layer and ending at the input layers. The mechanism for gradient propagation is based on the chain rule of differentiation.

Neural networks are sensitive to initialization of weights and preprocessing of the
inputs. The inputs are recommended to be normalized by subtracting the means and dividing by the standard deviations. The recommendation is to initialize the weights with random values close to zero (for example in the interval $[-0.7, 0.7]$). The sigmoid function is well-approximated by a linear function when its input is close to 0. This means that initially the network implements a linear function. Early stopping is recommended to avoid overfitting. The interpretation of that rule is that since the network starts as a linear function when training stops early, the network remains close to a linear function.

2.2.4 Boosting

The Ada-boost learning algorithm solves classification problems through a linear combination of weak learners. A weak learner is a classification algorithm that has an error rate slightly better than random. Boosting operates by fitting consecutively weak classifiers on re-weighted sets of data. After the algorithm learns a weak classifier, it reweights the data. The outcome of that process is that examples which were misclassified have their weights increased, while examples that are correctly classified have their weights decreased. The weak classifiers are decision stumps or very short trees. The final classifier is a weighted linear combination of decision stumps. Boosting has been proven to be very effective in practice. Analysis of boosting has revealed that it minimizes greedily an approximation to the error rate [16].

2.2.5 Support vector machine

Similar to logistic regression and Ada-boost, support vector machines (SVM) are used for binary classification problems. SVM are most easily defined on separable data and then the solution is extended to the case where perfect separability between the classes is not possible. In the separable case the classes are separated by a linear decision boundary. The margin is defined as the smallest distance between any of the points to the decision boundary. One should prefer classifiers in which the margin is the largest because they are the most stable ones with respect
to changes in the dataset. The points whose distance to the decision boundary is equal to the margin are called the support vectors of the classifier. The formal optimization problem solved by SVM is:

\[
\begin{align*}
\text{minimize:} & \; w, b \|w\|^2 \text{ (note: } \frac{1}{\|w\|} : \text{ the margin) } \\
\text{subject to:} & \; y_i (w^T x_i + b) \geq 1 \text{ (note: separation conditions) }
\end{align*}
\]

Given that perfect separation is often not possible, one imagine sliding the points that lie on the wrong side of the decision boundary on a hyperplane perpendicular to the decision boundary until perfect separation becomes possible. This idea corresponds to relaxation of the original optimization problem by the introduction of slack variables \(\xi_i\):

\[
\begin{align*}
\text{minimize:} & \; w, b \|w\|^2 + C \sum_i \xi_i \\
\text{subject to:} & \; y_i (w^T x_i + b) \geq 1 - \xi_i
\end{align*}
\]

Using the technique of Lagrange multiplier the above formulation is equivalent to:

\[
\begin{align*}
\text{maximize:} & \; \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i^T x_j \\
\text{subject to:} & \; 0 \leq \alpha_i \leq C, i = 1 \ldots N \\
& \; \sum_{i=1}^{N} \alpha_i y_i = 0
\end{align*}
\]

One particular advantage of the formulation of in Equation 2.16 is that it allows for a generalization of SVM to non-linear classifiers. The solution to the SVM problem according to Equation 2.16 depends on the dot product between the feature vectors. If the feature vectors had been mapped into a new space by the mapping \(x_i \rightarrow \phi(x_i)\), then to solve the SVM problem in the new space, it suffices to know the dot product in that space. The dot product in the new space
Chapter 2. Information Retrieval Methods and Learning to Rank Algorithms

is given by the kernel function $k(x_i, x_j) = \phi(x_i)^T \phi(x_j)$.

SVM can also be formulated as a penalization method using the hinge loss function [17]:

$$\min_{w, b} L(y_i, x_i) + \lambda \|w\|^2$$

(2.17)

$L(y, x) = \max(0, 1 - y(w^T x + b))$ is the hinge loss function (2.18)

This formulation is important since it makes it possible to relate SVM to other learning methods, in particular logistic regression.

2.2.6 Learning to rank algorithms

Learning to rank represents the use the machine learning techniques to the problem of learning a function that is able to rank objects. In the field of Information Retrieval, the objects which one needs to rank are documents returned by a search engine.

Some traditional machine learning techniques such as classification are difficult to apply to ranking because the data in search problems is highly imbalanced. Typically IR datasets have only a few relevant documents and a large number of irrelevant ones. Attempts to minimize the error rate as most classifiers do, will result in trivial classifiers that classify most documents as non-relevant. More over, standard learning algorithms do not account that the data in ranking problems is on a per query basis. Improving predictions for points in the same query that are already ranked low is not useful. Instead, the algorithm should maximize the queries for which it can predict accurately just a relevant few documents.

The Information Retrieval community has considered multiple approaches to learning ranking functions that can be put into three broad classes: pointwise, pairwise and listwise approaches. Pointwise approaches model the score of the document directly. Among those approaches are standard approaches of classification and regression. The final ranking is obtained by sorting the predicted
scores. While those approaches are in general suboptimal, it was found that in the case of averaging multiple ranking models a pointwise model performs as well as other more complicated approaches \[18\]. Pointwise approaches may work even if queries contain only relevant or irrelevant documents as long as those types of documents are well represented in the dataset as a whole. Pairwise approaches, on the other hand, need each query to contain a sufficient number of relevant and non-relevant documents.

The **pairwise** approach has to do with the number of misordered pairs in a ranked list. Let documents \(i\) and \(j\) in some ranked list have labels \(r_i\) and \(r_j\) with \(r_i > r_j\). Let their ranks be \(\pi_i\) and \(\pi_j\). The pair \((i, j)\) in the ranked list is ordered incorrectly if \(\pi_i < \pi_j\). The models from the pairwise category consider learning a pointwise model of documents scores: \(s_i = f(x_i)\) but using a pairwise training criterion. The training criterion uses the difference between scores \(s_i - s_j\) in an attempt to minimize a quantity related to the number of misclassified pairs in the list. During the test phase the model produces scores for each document and ranking by scores is sufficient.

Theoretical justification of pairwise training has been given in \[19\] since the number of misclassified pairs can serve as a lower bound to average precision. Pairwise models can be considered as an approximation to optimization of the area of the ROC curve. Since in Information Retrieval the area under the ROC curve is not of direct interest, but rather the area under some portion of it with a small number of false positives, pairwise models could be improved upon. The pairwise models are influenced negatively in terms of performance by a large number of documents in the least relevant categories, e.g. category 0 and 1 as swapping two documents with grades 0 and 1 is treated as severely as swapping documents with grades 4 and 3.

**Listwise** models operate on the complete ranked list, rather than on individual pairs. Typically, they work iteratively by incrementally producing models that give improved ranked lists. The quality of a ranked list is measured by an Information Retrieval measure of interest such as NDCG, mean average precision or ERR. At
any iteration those models could consider local improvements such as swapping pairs of documents that will improve the ranking. One example in this category is SoftRank [20] which defines a smooth approximation to the NDCG cost function.

**RankNet**

In RankNet a neural network model of the document scores is considered: $s_i = f(x_i)$ where $f$ represents a neural network. Given two scores for documents $i$ and $j$, it is possible to model the probability that documents $i$ and $j$ are ordered incorrectly. Let document $i$ be more relevant than document $j$. Then the probability that the pair $(i, j)$ is ranked incorrectly is:

$$p_{ij} = \frac{1}{1 + \exp (s_i - s_j)} \tag{2.19}$$

If the pair is ordered correctly, then $s_i > s_j$. This implies $p_{ij} < 0.5$. An incorrectly ordered pair would have $s_i < s_j$ and by implication $p_{ij} > 0.5$.

RankNet works in two passes. In the first pass, RankNet assigns scores to each document by using the neural network $f$. It then ranks the documents for each query. For each pair of documents $i$ and $j$ such that $i$ is more relevant than $j$, the misclassification probability of the pair $p_{ij}$ is computed. For each document define the sets $P_{i>j}$ and $P_{i<j}$ consisting of documents $j$ that are respectively less relevant and more relevant than $i$:

Let $P_{i>j}$ be the set of all documents $j$ from the same query as document $i$, such that $\text{rel}_i > \text{rel}_j$

Let $P_{i<j}$ be the set of all documents $j$ from the same query as document $i$, such that $\text{rel}_i < \text{rel}_j$

Then define $p_i$ as the average of the corresponding $p_{ij}$:

$$p_i = \sum_{j \in S_i} p_{ij} - \sum_{j \in T_i} p_{ji} \tag{2.20}$$
The value of \( p_i \) acts as an average error for document \( i \). The value of \( p_i \) is treated as a gradient for each document \( i \) which has to be back-propagated through the network to adjust the parameters of the learning model.

The development of RankNet was based on two key ideas. The first is that a neural network can be used as a fitting mechanism. This idea is not critical to the ranking problem as other learning algorithms could also work with gradient information. The second idea is that the definition of the gradient should follow from the objective to minimize the number of swapped pairs between relevant and non-relevant documents.

The estimated probability of misordering a pair \( p_{ij} \) follows from the goal of minimization of the cross entropy between the model assigned probabilities \( p_{ij} \) of two document and the true ordering the pair. The details of RankNet algorithm are given in Figure 1.

**LambdaRank**

LambdaRank can be considered an improvement on RankNet by defining a gradient function which is more closely related to the goal of optimization of NDCG on a training set. The problem with most Information Retrieval measures is that they are non-smooth and non-differentiable. This is due to the fact that most Information Retrieval measures rely on sorting of the documents by their scores, which is a non-differentiable operation. One direct implication of this fact is that small changes in the model may not yield large enough changes in the objective. This in turn precludes many techniques that rely on local iterative improvements such as gradient descent. One possible way out of this problem is to change the training objective with a smooth differentiable function. LambdaRank sidesteps this problem by defining a derivative with respect to a pair of document scores directly. It can be argued that this is an easier problem because specification of the derivative would allow us to directly force a local improvement to the current ranking. Back-propagation finds out the required changes in the model in order to achieve the local improvement suggested by the gradient. The derivative of LambdaRank with respect to a pair of scores is a combination of the derivative of
Algorithm 1 RankNet and Lambda Rank Algorithms

**Initialization**
Let $\beta = \text{random vector with components close to 0}$

**Iterative Improvement Steps:**
while stopping condition not met do
    Forward pass:
    for each query do
        compute the score of each document: $s_i = f(x_i, \beta)$
        sort documents by score
    end for
    Backward pass:
    for each query do
        for each pair of documents in query with rel$_i >$ rel$_j$ do
            $d_{ij} = s_i - s_j$ is the difference between scores
            $p_{ij} = \frac{1}{1 + \exp(-d_{ij})}$ is the probability that $i$ is ranked after $j$
            $\pi_i$ is the rank of document $i$ when documents are sorted by score
            $\lambda_{ij} = p_{ij}$ For ranknet
            $\lambda_{ij} = \frac{1}{\max DCG} p_{ij} (2^{\text{rel}_i} - 2^{\text{rel}_j}) \text{abs}(\frac{1}{\log(\pi_i + 1)} - \frac{1}{\log(\pi_j + 1)})$ For LambdaRank
        end for
        Let $P_{i>j}$ be the set of all documents $j$ from the same query as document $i$, such that rel$_i >$ rel$_j$
        Let $P_{i<j}$ be the set of all documents $j$ from the same query as document $i$, such that rel$_i <$ rel$_j$
        $\lambda_{i>j} = \sum_{j \in P_{i>j}} \lambda_{ij}$
        $\lambda_{i<j} = \sum_{j \in P_{i<j}} \lambda_{ji}$
    end for
    Let $\nabla \beta = \sum_{\text{query}} \sum_{i \in \text{docs}} \text{back-propagate}(\lambda_{i>j} - \lambda_{i<j})$
    Parameter improvement:
    $\beta_{\text{new}} = \beta - \gamma \nabla \beta$
end while
RankNet and a weight factor implied by NDCG. If $p_{ij}$ is defined as in RankNet, then $\lambda_{ij}$ is used to obtain the derivative of the implicit cost $C$ with respect to the score of document $s_i$:

$$
\lambda_{ij} = \frac{1}{\text{maxDCG}} p_{ij} (2^{\text{rel}_i} - 2^{\text{rel}_j}) \text{abs} \left( \frac{1}{\log(\pi_i + 1)} - \frac{1}{\log(\pi_j + 1)} \right) (2.21)
$$
defined when $\text{rel}_i > \text{rel}_j$

$$
\frac{\partial C}{\partial s_i} = \lambda_{i>j} - \lambda_{i<j} = \sum_{j \in P_{i>j}} \lambda_{ij} - \sum_{j \in P_{i<j}} \lambda_{ji}
$$

Ranking SVM

Ranking SVM is another model based on document pairs. The formulation is equivalent to creating a new training set consisting of correctly ordered pairs and applying the standard SVM algorithm. Let $P_{i>j}$ denote the set of documents from multiple queries for which it is known that document $i$ is more relevant than document $j$. Then the problem solved by Ranking SVM is a relaxation of the problem of finding the coefficients $(\beta, \xi)$ that results in a minimum number of swapped pairs in the ranking on the training set:

$$
\text{minimize: } \frac{1}{2} \beta^T \beta + C \sum_{(i,j) \in P_{i>j}} \xi_{i,j}
$$

subject to:

$$
\beta^T \phi(x_i) \geq \beta^T \phi(x_j) + 1 - \xi_{i,j}, \forall (i, j) \in P_{i>j}
$$

$$
\xi_{i,j} \geq 0
$$

In practice, the minimum number of swapped pairs is computed for each query and then averaged across queries. The value of $\beta^T \beta$ in the optimization criterion acts as a regularization or penalty in the magnitude of the coefficients while the constant $C$ trades off the number of swapped pairs with the stability of the solution.
Rank Boost

Rank Boost is another algorithm from the pairwise paradigm derived by extension of the standard boosting algorithm for classification. Similar to RankNet and Ranking SVM, Rank Boost operates on pairs of training examples, but derives a ranking function that is able to score each test point separately. Rank Boost proceeds in rounds where after each round a new weak ranker is derived and a distribution over the pairs of training examples is maintained. The weak rankers are decision tree stumps chosen at every iteration to minimize greedily a bound on the number of swapped pairs. The distribution of weights over the training pairs is updated to emphasize swapped pairs for the next iteration. There is an approximation which approximates the quantity to be optimized by a function linear in the features:

\[
D(x_i, x_j) \exp(\alpha(h(x_i) - h(x_j)) \approx D(x_i, x_j)[h(x_i) - h(x_j)]
\]  \hspace{1cm} (2.22)

In the above equation \( h \) is used to denote the current hypothesis of the learner. This approximation makes possible an algorithm that is linear in the number of training examples, instead of the number of pairs. \( D(x_i, x_j) \) is the distribution weight assigned to the pair \( i, j \). For convenience \( D(x_i, x_j) \) is defined to be zero when \( \text{rel}_i < \text{rel}_j \). Pairs which had been misclassified in the previous iterations will have large weights in the next iteration.

Move over the linear approximation to the pairwise rankboost penalty is equal to:

\[
\sum_{i,j} D(x_i, x_j)(h(x_i) - h(x_j)) = \sum_i x_i d_i
\]  \hspace{1cm} (2.23)

where \( d_i = \sum_j (D(x_i, x_j) - D(x_i, x_j)) \)  \hspace{1cm} (2.24)

The function \( d_i \) is called the potential of the point \( i \) and gives the importance of the point in the training set.
2.2.7 Comparison of RankNet, Rank Boost and Ranking SVM

It is instructive to compare certain quantities that drive the progress in optimization in RankNet and Rank Boost algorithms. In Rank Boost the updates of parameters depend on the so called potential of each point: 

\[ d(x_i) = \sum_{j \in P_{i,j}} D(x_i, x_j) - \sum_{j \in P_{i,j}} D(x_j, x_i) \]

which, after the first round, is equal to:

\[ \sum_{j \in P_{i,j}} \frac{\exp(\alpha(h(x_i) - h(x_j)))}{Z} - \sum_{j \in P_{i,j}} \frac{\exp(\alpha(h(x_j) - h(x_i)))}{Z} \quad (2.25) \]

In the above equation \( Z \) is a normalization factor.

In RankNet parameter updates depend on the gradient, also called lambda, defined as:

\[ \sum_{j \in P_{i,j}} \frac{1}{1 + \exp(s_i - s_j)} - \sum_{j \in P_{i,j}} \frac{1}{1 + \exp(s_j - s_i)} \quad (2.26) \]

In Ranking SVM with linear kernel the importance of a training point can be deduced by writing the SVM problem as an unconstrained problem (see e.g. [21]), as in Equation 2.18. Computing the gradient of the SVM and grouping the contribution to the gradient by training points, one can see that the “importance of a training point” is given by:

\[ \sum_{j \in P_{i,j}} L(x_i, x_j) - \sum_{j \in P_{i,j}} L(x_j, x_i) \quad (2.27) \]

\[
L(x_i, x_j) = \begin{cases} 
1 & \text{if } i \text{ and } j \text{ are incorrectly ordered}(s_i < s_j) \\
0 & \text{otherwise}
\end{cases} \quad (2.28)
\]

All three models are related as they use functions that measure the losses due to incorrectly ordering pairs of documents. However, the three models define differently the pairwise loss.
LambdaMart

LambdaMart is the result of the combination of two ideas: (1) the gradients of LambdaMart and (2) the very flexible learning method called gradient boosted machine. The Gradient Boosted Machine learning method (GBM) was created by Jerome Friedman et al. and motivated by his earlier work exploring the connection between Boosting and Additive Logistic Regression [22]. GBM can be adapted to various loss functions, for example squared loss for classification, cross entropy loss for classification and even pairwise loss for ranking. GBM allows for the construction of flexible non-linear decision boundaries. The form of GBM is:

\[ F(x) = \sum_i f_i(x) \quad \text{score is additive linear combination} \quad (2.29) \]
\[ L(y, F(x)) \quad \text{is the loss function} \quad (2.30) \]
\[ f_i \quad \text{are weak learners} \quad (2.31) \]

The model \( F(x) \) is computed greedily. At each iteration a new function \( f_i(x) \) is trained. Let \( F_m = f_1(x) + \ldots + f_m(x) \) represents the partial model obtained at iteration \( m \). One way to proceed is to improve slightly the current solution by moving in the direction of the negated gradient:

\[
F_{m+1}(x) = F_m(x) - \gamma \frac{\partial L(y, F)}{\partial F} \bigg|_{F=F_m(x)} \quad (2.32)
\]

The gradient could be evaluated at every training point. However, during testing we want to evaluate the gradient at new points. The solution of GBM is to use machine learning techniques to learn the gradient itself in terms of feature values. GBM, therefore, reduces the problem of learning the \( m \) weak learner \( f_m \) to building a (shallow) regression tree which predicts the gradient of \( F_{m-1} \). For a regression tree, the predictions at the leaves all have the same value. Let \( c_k \) be the value that the regression tree should assign to all test points which reach that leaf. Let those values form the region \( R_k \) What is the optimal value of \( c_k \)? In some cases, for example squared error loss, a closed form solution is just the average of the
gradients. When closed form solution is not possible a Newton-Raphson update can be used:

\[
    c_k = \frac{\sum_{i \in R_k} g(x_i)}{\sum_{i \in R_k} g'(x_i)} 
\]

(2.33)

where \( g(x_i) = \frac{\partial L(y, F)}{\partial F} \bigg|_{F=F_m(x)} \)

(2.34)

and \( g'(x_i) = \frac{\partial L(y, F)}{\partial F \partial F} \bigg|_{F=F_m(x)} \)

(2.35)

To summarize, the GBM model at iteration \( m \) is:

\[
    F_{m+1}(x) = F_m(x) - \gamma \begin{cases} 
    c_1 & \text{when } x_i \in R_1 \\
    \ldots \\
    c_k & \text{when } x_i \in R_k 
\end{cases} 
\]

(2.37)

The parameter \( \gamma \) is the learning rate. The GBM algorithm makes use of a number of important considerations not mentioned here. Those include the choice of a random subset of the data to derive the weak learners, the choice of the learning rate and the number of regions in the computed tree. Reference [23] can be used to study those properties, while reference [24] describes in detail the implementation under various loss functions.

In LambdaMart \( g(x_i) \) is the LambdaRank gradient computed on pairs. In that
Case $c_k$ is equal to:

$$c_k = \frac{\epsilon + \sum_{s \in R_k} \lambda_s}{\epsilon + \sum_{s \in R_k} n_s}$$ (2.38)

where $p_{ij} = \text{sigmoid}(-(F(x_i) - F(x_j)))$ (2.39)

label-weight$_{ij} = (2^{rel_i} - 1) - (2^{rel_j} - 1)$ (2.40)

rank-weight$_{ij} = \text{abs} \left( \frac{1}{\log(1 + \pi_i)} - \frac{1}{\log(1 + \pi_j)} \right)$ (2.41)

$w_{ij} = \text{label-weight}_{ij} \times \text{rank-weight}_{ij}$ (2.42)

$\lambda_{ij} = \frac{1}{\max DCG} \times p_{ij} \times w_{ij}$ (2.43)

$\lambda_i = \sum_{j \in P_{i > j}} \lambda_{ij} - \sum_{j \in P_{i < j}} \lambda_{ji}$ (2.44)

$n_i = \sum_{j \in P_{i > j}} p_{ij}(1 - p_{ij})w_{ij} + \sum_{j \in P_{i < j}} p_{ij}(1 - p_{ij})w_{ij}$ (2.45)

The epsilon is according to the implementation in [25].

### 2.3 Classification of Learning Algorithms

In Table 2.1 I present a classification of some common learning algorithm by two important dimensions: (1) the loss function and (2) the concept/hypothesis space. In a number of cases, it is possible to use the same algorithm with different loss function. Learning to rank differs from other common machine learning tasks such as classification and regression by the choice of the loss function. Pairwise learning to rank algorithms use pairwise loss functions. The actual pairwise loss function are closely related to the loss functions used in classification. For example, SVM for classification uses the hinge loss, while SVM for ranking uses the same hinge loss for pairs of documents from different categories.

The concept/hypothesis space is the representation used to relate the features to the target label. Some common hypothesis spaces include (1) linear combination of features (2) linear combinations of non-linear transformations of the features and (3) tree-based methods.
3 The Theory of Optimal Design of Experiments

3.1 Introduction

The Optimal Design of Experiments (ODE) [26, 27] is a statistical theory that provides a framework for efficiently conducting experiments. The goal of an experiment, as defined by this theory, could be to estimate the coefficients of a model of a physical process. According to ODE, an experiment should be conducted under such conditions that the variance of an estimated variable is minimized. The optimal conditions under which the experiment should be performed are called optimal design. The goal of minimization of the variance could be modeled mathematically in various ways. There exist a large number of optimality criteria that can be targeted: A-optimal design, D-optimal design, etc. In machine learning, the models estimate coefficients based on data. The design in machine learning problems constitutes the data points which the learning method will use to estimate the coefficients.

The simplest case of ODE is multiple linear regression with uncorrelated errors with constant variance. More complex models follow from extensions of the theory for this case.

3.2 Optimal Design of Experiments for the Linear Model

Let us assume the typical machine learning setup of a dataset $X$ with labeled feature vectors. Let the label be related to the feature values (predictors) through a linear function:

$$y = x^T \beta + \epsilon$$  \hspace{1cm} (3.1)
where \( y \) is the label, \( x \) is a column vector of feature values, \( \beta \) is the set of model parameters that need to be obtained through fitting a dataset, \( \epsilon \) is noise.

The fitting is done on a training data set \((Y_{\text{train}}, X_{\text{train}})\) with \( X_{\text{train}} \) being an \( N \times F \) matrix (called the design matrix) that represents \( N \) training feature vectors each having \( F \) features and \( Y_{\text{train}} \) is a vector of outputs; \( \beta \) represents the true, but unknown vector of \( F \) coefficients that needs to be estimated by fitting the training data.

\[
Y_{\text{train}} = X_{\text{train}} \beta + \epsilon \tag{3.2}
\]

\[
\begin{pmatrix}
    y_1^T \\
    y_2^T \\
    \vdots \\
    y_N^T
\end{pmatrix};
X_{\text{train}} =
\begin{pmatrix}
    x_{11} & x_{12} & \ldots & x_{1F} \\
    x_{21} & x_{22} & \ldots & x_{2F} \\
    \vdots & \vdots & \ldots & \vdots \\
    x_{N1} & x_{N2} & \ldots & x_{NF}
\end{pmatrix}
\]

\[
\beta =
\begin{pmatrix}
    \beta_1 \\
    \beta_2 \\
    \vdots \\
    \beta_F
\end{pmatrix}
\]

\[
x_i^T =
\begin{pmatrix}
    x_{i1} & x_{i2} & \ldots & x_{iF}
\end{pmatrix}^T \text{ is a feature vector}
\]

The usual assumption under the linear model is that the noise \( \epsilon \) is a random variable with zero mean and constant variance and that the errors between any two observations are uncorrelated:

\[
E(\epsilon_i) = 0 \tag{3.3}
\]
\[
\text{var}(\epsilon_i) = \sigma^2 \tag{3.4}
\]
\[
\text{Cov}(\epsilon_i, \epsilon_j) = 0, \quad i \neq j \tag{3.5}
\]

In Statistics typically the data \( X_{\text{train}} \) is considered non-random and the only randomness comes from \( \epsilon \): the noise with which \( y \) is measured. It is because of the noise \( \epsilon \) that every time we take a new data \( X \) to estimate \( \beta \) we obtain a different answer. The following estimator \( \hat{\beta} \) is the minimum variance unbiased estimator of \( \beta \):

\[
\hat{\beta} = (X_{\text{train}}^T X_{\text{train}})^{-1} X_{\text{train}}^T Y_{\text{train}} \tag{3.6}
\]
In practice, unbiased estimates do not perform well on prediction tasks. One solution is to use ridge regression by minimization of the sum of squared errors and penalty on the magnitude of the coefficients: \[ \sum_{i=1}^{N} (y_i - x_i^T \beta)^2 + \lambda \sum_{i=1}^{K} \beta_i^2. \] Regularized regression shrinks the estimator of \( \beta_i \) towards 0 and the predictions \( \hat{y}_i \) towards the global mean \( \bar{y} \). By shrinking extreme values, the variance of the predictions is decreased, but the predicted labels are no longer unbiased estimates.\(^1\)

The expected error at a test point \((y_{\text{test}}, x_{\text{test}})\) can be decomposed into squared bias and variance:

\[
E[(y_{\text{test}} - \hat{y}_{\text{test}})^2] = \text{bias}_{x_{\text{test}}}^2 + \text{var}(y_{\text{test}})^2 \\
\text{var}(y_{\text{test}}) = \sigma^2 x_{\text{test}}^T (\lambda I + X_{\text{train}}^T X_{\text{train}})^{-1} x_{\text{test}}
\]

The squared bias is the square of the difference between the average prediction (over multiple training datasets) and the optimal prediction. The variance measures the fluctuation of individual predictions due to multiple datasets around the average prediction. The expected value of the noise is zero from the assumptions of the linear model. Minimization of the variance of the test points can result in minimization of the average test error, especially when the variance component dominates.

The bias and variance are traded-off through the parameter \( \lambda \). For example \( \lambda = 0 \) will make the bias zero, but would increase the variance.

According to ODE the “optimal dataset” can be defined as the one causing the smallest variance in the model parameter \( \beta \). For ridge regression the estimator \( \hat{\beta} \)

\(^1\)Unbiased estimates of \( y_i \) have in expectation the true label, but biased estimates are systematically off.
The matrix $X_{train}^T X_{train}$ is known as the information matrix. Regularization by ridge regression causes the matrix to be replaced by $\lambda I + X_{train}^T X_{train}$. I will call this matrix $M_N$. I use $N$ to emphasize that $M_N$ is computed from $N$ points. The information matrix $N$ is the summation of $N$ rank-1 information matrices $x_i x_i^T$. Each rank-1 matrix is defined for a single training point $x_i$. The inverse of the information matrix $M_N$ is related to the covariance matrix of the parameter estimates $\hat{\beta}$ (Equation 3.10). The covariance matrix can be used to put a "confidence interval" around the parameter estimates. The "confidence interval" is an $F$ dimensional ellipsoid ($F$ is number of features). The volume of this ellipsoid is a function of the determinant of the inverse information matrix. By making the determinant of $M_N^{-1}$ small, the model becomes more stable. The data selection criterion that maximizes the determinant of the information matrix is known as D-optimal design[26, 27]. The mathematical optimization problem can be stated as follows:

$$\hat{\beta} = (\lambda I + X_{train}^T X_{train})^{-1} X_{train}^T Y_{train}$$  \hspace{1cm} (3.9)$$

with \quad \text{Cov}(\hat{\beta}) = \sigma^2 (\lambda I + X_{train}^T X_{train})^{-1} \quad \text{covariance matrix} \hspace{1cm} (3.10)$$

and \quad \sum_{i=1}^{F} \text{var}(\hat{\beta}_i) = \sigma^2 \text{trace} \left( (\lambda I + X_{train}^T X_{train})^{-1} \right) \hspace{1cm} (3.11)$$
Choose \( R \) training points \( x_{i_1}, x_{i_2}, \ldots, x_{i_R} \) \((3.12)\)
from the training set \( X_{\text{train}} = (x_1, x_2, \ldots, x_N) \)
which maximize the determinant \( \det \left( \lambda I + X_{\text{sel}}^T X_{\text{sel}} \right) \)
where \( X_{\text{sel}} = \begin{pmatrix}
    x_i^T \\
    x_{i_2}^T \\
    \vdots \\
    x_{i_R}^T
\end{pmatrix} \) \((3.13)\)

This criterion maximizes the confidence in the model parameters, by minimizing
the confidence ellipsoid around the point estimate \( \hat{\beta} \). In the following, I perform
greedy optimization by adding point \( i_R \) to the set of previously selected \( i_R \) points.

Let \( M_i = \lambda I_F + X_{\text{sel}}^T X_{\text{sel}} \) be the information matrix (feature correlation matrix)
computed from the first \( i \) selected points. Additionally, define \( \tilde{M}_i = \lambda I_i + X_{\text{sel}} X_{\text{sel}}^T \)
be the “data correlation matrix” of size \( i \times i \) computed on the first \( i \) selected data
points.

The newly added training data point \( x_{i_R} \) leads to the addition of a rank-1
information matrix \( x_{i_R} x_{i_R}^T \) to \( M_{R-1} \):

\[
det(M_R) = \\
det(M_{R-1} + x_{i_R} x_{i_R}^T) = \\
det(M_{R-1})(1 + x_{i_R}^T M_{R-1}^{-1} x_{i_R}) = \\
det(M_{R-1})(1 + \text{var}(x_{i_R}|M_{R-1}))
\]

This above equation expresses that the best new point to be added to the existing
\( R - 1 \) points, is the one with largest variance given a linear model trained on the
first \( R - 1 \) points. Therefore, minimization of the variance of \( \beta \) leads indirectly to
minimization of the variance of test points and as a consequence minimization of
the error at a test point.

A different perspective of the D-optimal design can be obtained by considering the mathematical equality between the determinants of the covariance matrix and the data correlation matrix.

$$\det(M_R) = \det(\lambda I_F + X_{sel}^T X_{sel}) = \det(\lambda I_R + X_{sel}^T X_{sel}^T) = \det(\tilde{M}_R)$$

The newly added data point $x_{ir}$ is represented as a new row and column in the “data correlation matrix” $\tilde{M}_R$. Using the property of the determinant of subtracting one row from another one, I obtain:

$$\det(\tilde{M}_R) = \det(\tilde{M}_{R-1}) \times$$

$$(K(i_R,i_R) - Vec(i_R,i_1,...,i_{R-1})\tilde{M}_{R-1}^{-1}Vec(i_R,i_1,...,i_{R-1})^T)$$

where $Vec(i_R,i_1,...,i_{R-1}) = (K(i_1,i_R), \ldots, K(i_{R-1},i_R))$

and $K(i_t,i_s) = x_{it}^T x_{is} + \delta(i_t,i_s)$

$\delta$ is the Kronecker delta

The function $K(.,.)$ is a dot product between two feature vectors, while $Vec$ is vector with components the dot products of the new point and each of the previously added points. The interpretation of this result is that the best point to be added is the one which simultaneously has a large norm and is most orthogonal to the previously added points. Formula 3.16 arises in ODE for regression with correlated errors in the observations (kridging)[27]. The formula also appears in the context of Gaussian Processes and is related to the variance of a test point. In those settings, the function $K(.,.)$ is not used specifically as a dot product, but can denote any kernel. Interpreting somewhat freely the kernel function $K(.,.)$ as a similarity between two points one could argue that the new point to be added is most “dissimilar” to the previously selected points.
3.3 Illustration of D-optimal Designs

It is helpful to visualize the selected points in the D-optimal design for a two-dimensional problem. In figure 3.1 I have generated a dataset using a two-dimensional Gaussian distribution. The data looks spherical and the points in the design appear at the boundary of the ellipsoid. The points were selected by a greedy algorithm that repeatedly chooses to add the point which causes the greatest increase in the determinant of the information matrix.

Figure 3.2 shows the effect of different values of $\lambda$. When $\lambda$ is smaller the angle between the closest points in the design is emphasized more than the length of the vector representing the points. This effect can be deduced from Equation 3.16 where the first component deals with the length of the feature vector, while the second deals with dot product between the selected point and the previous points.

As it can be observed from the plot in Figure 3.2, the value of $\lambda$ is related to the relative emphasis on each feature. Large $\lambda$ will diminish the effect of features with small variances. As a rule of thumb, features with larger variances are more informative, but without apriori knowledge the experimenter may wish to treat all features equally.
3.4 Relationship of ODE to Generalization Error

The generalization error is defined as the expected error to be observed on a new test set. In some cases, the leaving-one-out (L-O-O) method allows analytic approximation of the generalization error. The L-O-O method computes the generalization error by using the training set. For a training set of \( N \) points, the L-O-O method creates \( N \) virtual training sets where each of these sets has \( N - 1 \) training points. The skipped point is used as a test point and the error on this point is measured. The L-O-O error is the average of the error computed on \( N \) test points. For linear regression, a closed form approximation of L-O-O error called generalized cross validation (GCV) is available [28, 17]:

\[
\sum_{i=1}^{N} \left[ \frac{(y_i - \hat{y}_i)^2}{N - \text{trace}(S)} \right] \\
\text{where } S = X(X^TX + \lambda I)^{-1}X
\]

Figure 3.2: Demonstration of the effect of \( \lambda \) for D-optimal design in linear regression.
The numerator is related to the mean squared error on the training set, while the denominator is equal to the trace of the inverse of the information matrix:

$$\text{tr} \left[ (X^T X + \lambda I)^{-1} \right]$$  \hspace{1cm} (3.19)

The above Equation 3.19 is the criterion for A-optimal design from ODE which measures the sum of variances of the model coefficients.

To minimize the value of Formula 3.18, we can choose the points which bring the largest contribution. The numerators depend on each individual point. Therefore, we should prefer the points for training which have large errors. On the other hand, the denominator depends on the complete training set. It is the sum of the variances of all training points. For regression, the value is largest for points with large feature vectors and positioned in regions which have the least number of points. Similar to the D-optimality criterion in ODE, the GCV error is influenced by two components (1) local component related to the error at a training point and (2) global component related to the global configuration of feature vectors.

### 3.5 Optimization Algorithm

The most popular optimization algorithms for D-optimal designs are greedy. The following strategies can be used: greedily adding points, greedily deleting points and swapping points if the swap guarantees improvement of the criterion. I show the algorithm for greedily adding points. Given previously selected $k$ points $x_1, x_2, \ldots, x_k$ we wish to find $x_{\text{new}}$ which maximizes:

$$\text{det}(\lambda I + \sum_{i=1}^{k} x_i x_i^T + x_{\text{new}} x_{\text{new}}^T) = \text{det}(M_k + x_{\text{new}} x_{\text{new}}^T)$$  \hspace{1cm} (3.20)

The above is equal to the product of where the second determinant is of a scalar:

$$\text{det}(M_k + x_{\text{new}} x_{\text{new}}^T) = \text{det}(M_k) \text{det}(1 + x_{\text{new}}^T M_k^{-1} x_{\text{new}})$$  \hspace{1cm} (3.21)
$x_{\text{new}}^T M_k^{-1} x_{\text{new}}$ is the variance of the point when the model is trained on the previous $k$ points. Folding in $x_{\text{new}}$ into $M_k^{-1}$ can also be done efficiently using application of the Woodbury identity:

Let $b = M_k^{-1} x_{\text{new}}$

and $c = x_{\text{new}}^T M_k^{-1} x_{\text{new}}$

$$M_{k+1}^{-1} = M_k^{-1} - \frac{1}{1+c} bb^T$$

For efficiency reasons one can keep track of the products $M^{-1}x_i$. To greedily select $K$ points out of a dataset of $N$ points the running time is $O(KN)$.

### 3.6 Non-linear Models

In practice many interesting models are non-linear. One could differentiate between non-linearity of the model in the feature combinations and non-linearity of the link function. For example, logistic regression uses linear combination of features, but a non-linear link function. On the contrary, neural networks make use of non-linear combination of features.

I overview the application of ODE to logistic regression since it is helpful to illustrate how the optimal design changes when the optimized target function of the model changes. Then I discuss how ODE applies to non-linear feature transformations in the context of neural networks and Gaussian processes.

#### 3.6.1 Logistic regression

Logistic regression is a binary classification model that I described in Chapter 2. One common approach for estimation of the parameters in logistic regression is the iteratively least squares algorithm (IRLS). In IRLS one solves repeatedly a sequence of least squares problems in which the features and the labels are modified at each iteration. Since the information matrix depends on the model parameter $\beta$, the optimal design cannot be estimated without knowing $\beta$. The approach of
adaptive designs for dataset selection in non-linear models [27] solves this problem by alternating data selection with estimation. If the optimal parameter $\beta$ were known, it could be used to determine the optimal design. If the design was known, it would be used to estimate $\beta$.

The solution of the D-optimal design for logistic regression is dictated by the information matrix:

\[
\text{choose data to maximize the determinant of the information matrix} \\
\det \left( \sum_{j=1}^{N} \text{obs. inf. matrix}(x_i) \right) \\
\text{obs. inf. matrix}(x_i) = s_i n_i d_i \text{ is the information matrix of observation } i \\
d_i = x_i x_i^T \text{ results in selected feature vectors being spread apart} \\
s_i = (p_i(1-p_i))^2 \text{ is the sensitivity due to the logistic function} \\
n_i = \frac{1}{p_i(1-p_i)} \text{ is the inverse of noise caused by a binomial random variable} \\
\text{where } p_i = \frac{1}{1 + \exp(\beta^T x_i)} \text{ is the estimated probability of } i \text{ to be positive example}
\]

As it can be seen a number two factors play a role in the solution: the dot product between the feature vectors and the probability of class membership. The product between the feature vectors is similar to the case of the linear model. However, for logistic regression certain points are emphasized through the use of $p_i$. The logistic regression function is most sensitive in the region close to the decision boundary ($p_i \approx 0.5$) and therefore points in this region are emphasized. However, the variance due to the binomial random variable that is estimated (the noise) is also largest there. The trade-off is achieved by multiplying both factors $s_i$ (sensitivity) and $n_i$ (inverse noise).

The interpretation of Formula 3.22 is that in the optimal design:

• regions with noise small are preferred (provided noise depends on $x$ or $y$);
• the representation of the dataset is maximized;
• data should be selected from regions where the modeling criteria of interest is most sensitive.
The optimal design for logistic regression is illustrated on Figure 3.3 using data generated by a two-dimensional Gaussian distribution. The data has two classes: positive represented by green dots and negative represented by red triangle. Adaptive designs work iteratively. In the first iteration the selected points in the design are the same as in the case of the linear model. The second iteration uses the data from the first iteration to estimate $\beta$. Based on the estimate a new set of points is suggested. As it can be seen the new points are both (1) far from each other and (2) closer to the decision boundary. From the figure one can see how the points selected at every iteration of the adaptive design algorithm close in on the decision boundary. The distribution of the estimated probabilities $p_i$ affects how close the selected points would be to the decision boundary. The sigmoid in logistic regression can be written with an additional parameter $\gamma$:

$$p_i = \frac{1}{1 + \exp(\gamma \beta^T x_i)}$$ (3.22)

Even if we knew the optimal $\beta$ by changing $\gamma$ we can obtain quite different designs. For very small $\gamma$, the design approximates the D-optimal design for the linear model as the sigmoid is approximated well by a linear function in that case. For large $\gamma$, the values of $p$ for the points furthest from the decision boundary are very close to 0 or 1. By implication the weight $p(1 - p)$ of those points is close to 0. So, for large $\gamma$ only the points close to the decision boundary are effectively the ones considered in the design.

The following argument shows why choosing points only on the decision boundary is risky. Consider for example, one dimensional logistic regression. Suppose we choose two points on the decision boundary, so that $p$ for each of those points will be close to 0.5. The learning algorithm will be able to use this dataset only if those two points have different labels (and the labels should not be swapped). The probability of the points having correct labels is 0.25. As an alternative design consider the two points further from the decision boundary such that each has probability 0.7 of the correct label. Then the probability that both points have the correct label is 0.49. Creating the first dataset with points close to the decision
boundary and then choosing a label will result in the optimal decision boundary with probability 0.25. The second dataset may find a decision boundary a little bit far from the optimal, but this will happen with probability 0.5. The trade-off here is that a less accurate decision boundary can be found but with larger probability. This example also showed the effect of noise. When the positive and negative examples overlap, the decision boundary is noisy. The greater the noise, the further the optimal training examples are from the decision boundary.

### 3.6.2 Neural networks

Neural networks can be described as layers of functional transformations from the input to the desired output:

\[
\hat{y} = f(x, \beta)
\]  

(3.23)
Two common criteria in training are minimization of the squared error for regression or the cross entropy for classification. The optimization of the training criterion happens through the gradient of the error criterion with respect to the parameters. The gradient reflects how small changes in the parameters affect the final output. For efficiency, the gradient is computed backwards, by a procedure known as error backpropagation. The backpropagation procedure informs how small changes in the output are caused by changes in the parameters. The Hessian, denoted as $H$, is the matrix of second derivatives of the error function with respect to the parameters. For $M$ parameters, the Hessian is an $M$ by $M$ matrix with entries $\frac{\partial E}{\partial \beta_i \partial \beta_j}$, where $E$ is the optimization criterion that depends on the predicted and the true values of $y$. In nonlinear models the Hessian plays the same role as the information matrix $X^TX$ in the linear models. In order to obtain variance in the model parameters of a neural network one needs a probabilistic model. One such model is provided by the so called Bayesian Neural Network [29]. Assume the true value is distributed according to a Gaussian distribution around the predicted value:

$$p(t|x, \beta, \lambda_1) = N(y(x, \beta), \frac{1}{\lambda_1})$$

(3.24)

Assume a prior distribution of $\beta$:

$$p(\beta | \lambda_2) = N(0, \frac{1}{\lambda_2} I)$$

(3.25)

The log of the posterior distribution given the data is:

$$\log(p(w|D)) \approx -\lambda_2 \beta^T \beta - \lambda_1 \sum_{i=1}^{N} (y(x_i, \beta) - t_n)^2$$

(3.26)

Assume $\beta_{MAP}$ achieves (local) maximum of the above. The distribution of $\beta_{MAP}$ is approximated by a Gaussian whose covariance matrix is given by:

$$A = \lambda_2 I + \lambda_1 H$$

(3.27)
Therefore the data that results in parameters of the neural network with low variance, maximizes the determinant of the matrix $H + \lambda I$ (for suitable $\lambda$).

Computation of the Hessian of a neural network of $F$ parameters can be done in $F^2$ time using the technique of backpropagation. However, the simple greedy algorithm for computing the training set collection is no longer available. The Hessian has an approximation that allows the use of the same algorithm as in the case of the linear model:

$$H = \sum_{i=1}^{N} \frac{\partial y(x_i, \beta)}{\partial \beta} T \frac{\partial y(x_i, \beta)}{\partial \beta}$$ (3.28)

The Hessian is approximated by the outer product of first-order derivatives. This approximation is correct if the off-diagonal elements of the Hessian are close to zero.

A different approach which arrives at the same rule, is to linearize the neural network with a first-order Taylor expansion around a previous solution $\beta_0$:

$$y(x, \beta) \approx y(x, \beta_0) + (\beta - \beta_0)^T \frac{\partial y(x, \beta)}{\partial \beta} \bigg|_{\beta = \beta_0}$$ (3.29)

The above equation is linear in $(\beta - \beta_0)$ and applying the theory for the linear model, we compute the information matrix and find out that it is equal to the off-diagonal approximation of the Hessian in Equation 3.28. In effect, after linearization of the non-linear model, the partial derivatives play the effect of transformed features.

### 3.6.3 Gaussian processes

Gaussian Processes can be motivated by linear regression on transformed features. Let $x_i$ be the original $F$ dimensional features vectors which are transformed non-linearly to a feature vector with much larger dimension $M$:

$$x_i \rightarrow \phi(x_i)$$ (3.30)
One can write down a linear regression on the transformed features:

\[ y = \beta^T \phi(x) \]  

(3.31)

Let us use the least squared error criterion with quadratic penalty as an optimization criterion to solve for \( \beta^T \). Taking derivatives with respect to \( \beta \), we can write down the equation that \( \beta \) needs to satisfy:

\[ \beta = \frac{1}{\lambda} \sum_i \left( \beta^T \phi(x_i) - y_i \right) \phi(x_n) \]  

(3.32)

Let \( \Phi \) denote the design matrix of the transformed features. Introduce the vector \( \alpha = [\alpha_1, \ldots, \alpha_n]^T \) with components:

\[ \alpha_i = \frac{1}{\lambda} \left( \beta^T \phi(x_i) - y_i \right) \]  

(3.33)

Equations 3.32 and 3.33 represent two relationships between \( \beta \) and \( \alpha \) that can be solved for \( \alpha \) and \( \beta \):

\[ \alpha = (\Phi^T \Phi + \lambda I)y \]
\[ \beta = \Phi^T \alpha \]

The predictions for a new test point \( x \) are given by:

\[ y(x) = k(x)(\Phi \Phi^T + \lambda I)^{-1}y = k(x)(K + \lambda I)^{-1}y \]
\[ k(x) = [\phi(x)^T \phi(x_1), \ldots, \phi(x)^T \phi(x_n)]^T \]

The matrix \( K = \Phi \Phi^T \) has entries \( k(x_i, x_j) = \phi(x_i)^T \phi(x_j) \). The function \( k(x_i, x_j) \) is known as the kernel function and gives the influence (correlation) between two points. As can be seen from 3.34 the solution to the Gaussian Process depends on the dot product between the transformed feature vectors, just like SVM. One difference is that the predictions on an unseen point in SVM will depend only on a number of points, the support vectors in the SVM. For Gaussian processes the
predictions depend on all points.

Applying the theory of ODE to GP, one can see how Equation 3.15 can be generalized:

\[
det(\phi(X)^T W_N \phi(X) + I_M) = \\
det(W_N)det(\phi(X)\phi(X)^T + W_N^{-1}) = \\
det(K + W_N^{-1})
\]

where \(W_N = \text{diag}(w_1, \ldots, w_N)\) is a diagonal matrix of weights.

The above equation shows that the D-optimal dataset for GP will depend on the selection of points that are (1) non-similar because of the kernel function \(k(\cdot, \cdot)\) and on weights of the points \(W_N\) which (2) emphasize points important for the optimization criterion. The weights can be motivated by the use of a method such as logistic regression in kernel space.

### 3.7 Active Learning

Active Learning is a subfield of Machine Learning concerned with the “clever” selection of training data for machine learning algorithms. The idea is that if the learner is able to participate in the training process, then the number of training points required to achieve a desired accuracy will be smaller.

Active learning [30] interleaves over many iterations the steps of data selection and model update: new data points are selected based on the current model, and a new model is created based on the current dataset.

The Optimal Design of Experiments, described above, strives to minimize the number of training examples as well. In fact, it has been used by Machine Learning researchers as a paradigm in Active Learning [30]. For example, [31] shows an approximation to the A-optimality criterion for logistic regression. In the following,
I will describe known approaches for active learning and difficulties in applying active learning strategies for training collection construction.

Typically active learning happens one training point at a time. A training point is selected, it is labeled by a human judge and added to the model. The feedback from the selected point is used to update the model and propose the next training point.

**Strategies in Active Learning**

Common strategies for active learning include Uncertainty Sampling, Query By Committee and Error Reduction [30]. Under uncertainty sampling, one should select the training point about whose label, the learner is most uncertain. Under a two class classification problem, this is the point whose predicted label has probability of being in the positive class closest to 0.5. For multi-class classification problems, the entropy over the predicted distribution is often used. In regression settings, an instantiation of the uncertainty sampling could be the rule to choose the point with largest variance in the label and this is essentially the Optimal Design of Experiments approach.

Query By Committee (QBC) works by keeping a set ("committee") of competing hypotheses. The new point added to the training set is the one for which the members of the committee disagree the most. In [32] the committee is not generated explicitly. Instead access to the committee is provided via Gibbs sampling which generates committee members according to a posterior distribution over all possible hypotheses consistent with the currently generated training set. The theoretical analysis in [32] yields that a logarithmic number of examples in terms of the achieved error is needed. The success of this QBC method is attributed to its ability to reduce exponentially the number of hypotheses consistent with the data but the approach assumes noiseless data.

Application of QBC depends on the ability to sample directly from the posterior probability over the models. An algorithm called Hybrid Monte Carlo [33] allows sampling from the posterior distribution of the parameters of a neural network. In the case when sampling from the posterior is not possible, the ActiveDec-
orate [34] model can be useful. At every stage the model explicitly generates a committee of diverse classifiers by perturbing the labels and selects the training point with largest disagreement.

Under the expected error reduction framework [35], the goal is to select the example which reduces the error rate the most over a test collection. At a conceptual level, the method works by taking a possible training point and checking how adding it to the dataset changes the expected error. Before adding the point to the dataset, its label should be known. The current classifier is used to obtain a distribution of possible labels. Because each label is possible with some probability, we can obtain an expected error rate over the labels.

### 3.7.1 Active learning for learning to rank

Active learning has been applied to learning to rank, but the number of publications is quite small.

One of the earliest approaches on active selection of documents for learning is the Hedge algorithm [36]. The Hedge algorithm is actually a meta search algorithm rather than a learning algorithm. Hedge learns the optimal way to linearly combine pre-trained search algorithms rather than learning how to combine raw features. Hedge works in rounds where in each round it adds the document with maximum expected relevance. The weights of the systems which “voted” for this document are increased if the document did turn out to be relevant and decreased otherwise.

The work Donmez et al. [37] applies active learning for Ranking SVM with linear kernel and RankBoost. The idea there is to extend the training set by a document which is likely to cause the greatest change in the current model parameters. Ranking SVM can be written as an unconstrained optimization problem
using the hinge loss function:

\[
\min_{\beta} \sum_{(i,j) \in P} L(\text{sign}(y_i - y_j), (x_i - x_j)) + \lambda \beta^T \beta \quad (3.35)
\]

where \( L(z, v) = \max(0, 1 - z(\beta^T v)) \) is the hinge loss function (3.36)

\( z \): label of pair (3.37)

\( v \): difference between feature vectors (3.38)

\( \beta \): SVM model parameters (3.39)

\( P \) is the set of all pairs between documents with different labels (3.40)

Under the above formulation, it is possible to obtain the derivative of the model w.r.t \( \beta \) and factor out the contribution in the derivative for each training point.

For a new training point \( x_{new} \), the derivative depends on the difference between the feature vectors representing the new training point, and each of the old points \( x_j \). The derivative at the new point \( x_{new} \) is defined as:

\[
\frac{\partial y(x_{new}, \beta)}{\partial \beta} = \sum_{j \in T} \text{error}(new, j) \text{sign}(rel_{x_{new}} - rel_j)(x_{new} - x_j) \quad (3.41)
\]

\( T \): previous training set (3.42)

\[
\text{error}(new, j) = I[(rel_{x_{new}} - rel_j)(s_{new} - s_{j}) < 0] \quad (3.43)
\]

The new training point is compared to every point in the previously available training set which has different label than the new point. If the new point and the training point are swapped incorrectly in the ranking, then there is a contribution of \( \pm (x_{new} - x_j) \) to the derivative where the sign depends on the relevance labels of both points. To detect an error in the ranking of the pair, one checks whether rankings of documents \( new \) and \( j \) disagree according to their labels (\( rel_{new} \) and \( rel_j \)) and their scores (\( s_{new} \) and \( s_{j} \)). When the label of the new point is not available, it can be predicted. Let \( p_{new} \) be the probability that the new point is relevant. First, suppose that \( rel_j = 0 \). Then \( \text{error}(new, j) = 1 \) if \( s_{new} < s_{j} \) with probability \( p_{new} \). When \( rel_j = 0 \), \( \text{error}(new, j) = 1 \) if \( s_{new} > s_{j} \). Equations 3.43
becomes:

\[
\frac{\partial y(x_{\text{new}}, \beta)}{\partial \beta} = \sum_{j \in S_{\text{rel}} = 0} p_{\text{new}} I(s_{\text{new}} < s_j)(x_{\text{new}} - x_j) - \sum_{j \in S_{\text{rel}} = 1} (1 - p_{\text{new}}) I(s_{\text{new}} > s_j)(x_{\text{new}} - x_j) \tag{3.44}
\]

In the above, \( S_{\text{rel}} = k \) is the set of all points in the training set with relevance label \( k \), while \( I \) denotes the indicator function.

The authors in [37] define the change of the model as the expected sum of changes caused by incorrectly swapping the new point with each of the old points:

\[
\frac{\partial y(x_{\text{new}}, \beta)}{\partial \beta} = \sum_{j \in S_{\text{rel}} = 0} p_{\text{new}} I(s_{\text{new}} < s_j)\|x_{\text{new}} - x_j\| + \sum_{j \in S_{\text{rel}} = 1} (1 - p_{\text{new}}) I(s_{\text{new}} > s_j)\|x_{\text{new}} - x_j\| \tag{3.46}
\]

where \( \|v\| = v^T v \) is the norm of vector \( v \) \( \tag{3.47} \)

An approach for active learning in learning to rank problems called ELO is presented in [38]. That approach is based on the expected reduction of NDCG, after the addition of a new query or document to the training set.

It is easiest to explain their approach by example. Imagine a query with three documents and two ranking models. The ranking models are obtained by subsampling the available training set. Each model is used to provide scores to new unlabeled documents. The score \( s \) of a document is transformed through the NDCG gain function \( G_s = 2^s - 1 \). To obtain the NDCG of a ranking, the gains are discounted depending on the ranks as usual. Let the NDCG under both models be \( NDCG_1 \) and \( NDCG_2 \). The gains of the scores can be averaged and the NDCG can be computed again. Let us call it \( NDCG_{1+2} \). Then the value of interest is \( D = 0.5(NDCG_1 + NDCG_2) - NDCG_{1+2} \). A worked out example appears in Table 3.1

The value \( D \) expresses the difference between the average NDCG of the models
Table 3.1: Illustration of the method ELO for active learning in learning to rank after [38].

and the NDCG of the average model. If the rankings under both models are significantly different, then the ranking after averaging the gains will be significantly different from both rankings. This situation will result in large difference between the average NDCG and the NDCG of the average, making the query appealing for selection in the training set. Conversely, if both rankings were the same, the difference would be zero, and one would have the least incentive to use this query for training.

### 3.8 Unsuitability of Active Learning for Training Collection Construction

Given that active learning methods produce a training dataset while iteratively improving a learning model it might seem that active learning methods could be used for as training collections construction methods. There is some practical evidence that this is not the case. It is also possible to give theoretical arguments why active learning would not be suitable for training collections construction.

#### 3.8.1 Practical evidence

The study by Aslam et al. [8] compared data collections for learning to rank constructed by the Hedge algorithm with other collections construction methods. It was found out that while Hedge excelled at the machine learning task and
acquired the largest number of relevant documents, it was the weakest method for constructing training collections that are subsequently used by other learning algorithms.

The work of [39] constructed a dataset by an active learning method by choosing unlabeled data that estimated expected classification error of a newly proposed algorithm based on a Gaussian random field model. The constructed dataset was used to train and evaluate an SVM algorithm with a linear kernel. While the active learning method performed extremely well on the learning task the SVM method performed worse on the same dataset that it would have on a random dataset.

Those two examples show the difficulties in the use of data labeled in the process of an active learning method as a standalone dataset.

3.8.2 Theoretical arguments

One caveat against the usage of active learning for collection construction is that active learning explores too eagerly the properties of the learning algorithm under which it operates. The \( n \)-th training point that an active learner selects is optimal only given the previous \((n - 1)\) points. In fact, the \( n \)-th point is used to fix errors in the model which are due to the previous \((n - 1)\) points. Therefore, there is a path dependency in the learned model due to the data selection. It is not even clear if the same algorithm can be retrained with the same accuracy by removing information about the order in which points are selected. Consider a learning model that iteratively updates the parameter estimates \( \beta \). In active learning, the model produces the first estimate of \( \beta_0 \) using only a few points. The next estimate \( \beta_1 \) depends on \( \beta_0 \) and the points selected based on \( \beta_0 \). When the dataset compiled under the active learning procedure is used to estimate a new model, the sequence of estimates \( \beta_0, \beta_1, \ldots \) will depend on the complete dataset and will in general be different than the active learning solution.

The data selection under active learning is tightly coupled with the given learn-
ing algorithm. For example, optimal data selection for SVM with Gaussian kernel is provably non-optimal for training SVM with linear kernel. The theory of ODE can be used to prove this argument while the example by [39] referred to above, has shown that it does happen in practice.

Active learning is also susceptible to a problem known as sampling bias [40]. This name refers to the phenomenon that depending on the starting set of points active learners may fail to choose a point in a region of potential interest. The problem is caused by the lack of an appropriate method to estimate recall because of “unexplored parts of the space that contain positive examples”. Precision oriented tasks like ranking in Information Retrieval are less affected by those conclusions. The work of [41] proposed hierarchical active learning to avoid this issue.
4 Empirical Studies of the Properties of Datasets for Learning to Rank

The question that this chapter aims to answer is whether a dataset possesses intrinsic features that are predictive of its quality for a number of algorithms. Such properties are likely to be useful in the development and evaluation of training collections.

There are at least three aspects to the dataset worthy of consideration:

- (a) the distribution of document relevance labels;
- (b) feature properties, such as the diversity among feature vectors;
- (c) the number of queries vs. the number of documents in the training set.

4.1 Effect of the Distribution of Labels across Different Grades of Relevance in the Training Set

In this section I describe the results of a large-scale study that examines the effect of the distribution of labels across the different grades of relevance in the training set on the performance of trained ranking functions. In a controlled experiment I generate a large number of training datasets with different label distributions and employ three learning to rank algorithms over these datasets.

A post-hoc analysis of the effectiveness of a number of proposed training collection construction methods in [8] revealed that the ratio of relevant to non-relevant documents in the constructed dataset is one of the most influential factors. The dataset used by Aslam et al. was based on TREC 6, 7 and 8 ad-hoc corpora. It contained 150 queries with binary labels and 22 features per query-document pair.
Hence, even though this dataset allowed Aslam et al. to quantify performance as a function of the constructed training sets, the conclusions remain limited to small TREC collections and text-derived query features. Further, the construction methodology of the datasets did not allow Aslam et al. to observe the performance of ranking functions over the entire range of relevant/non-relevant ratios. Here I revisit the effect of the distribution of “positive” and “negative” examples in the training set by means of a large-scale study. In a controlled experiment, I construct a large number of training sets over the entire possible range of “positive” and “negative” instances and employ different learning to rank algorithms over these datasets. Understanding how the distribution of labels in the training set affects the quality of the trained ranking functions can provide a better insight into how researchers and practitioners should construct effective and efficient training data.

In this work I use the WEB30k dataset provided by Microsoft Research. This dataset comprises more than 30,000 user-issued web queries. For each query-url pair, 136 features, widely used in the research community, are provided. Relevance judgments (labels) are assigned according to a 5-grades scale, from 0 (completely irrelevant) to 4 (perfectly relevant). In order to control the distribution of positive and negative instances in the training set and given that the total number of labeled urls per query and the distribution of labels across the 5 relevance grades varies widely across the 30K queries, we only select those queries that have at least k labeled urls in each of the relevance grades. Further, we combine the two highest relevance grades into a single one, since typically labels in these grades are sparse in the collection. For k = 16 there are 655 queries that have at least 16 judged urls in each of the four grades, while for k = 8 there are 3676 such queries and so on. For each one of these datasets we randomly select 3/5 of the queries to construct the training set, 1/5 the validation and 1/5 the test set. There is no further processing of the test set.

For the training and validation sets I first select a total number of judgments per query n, with $n \leq k$. In the case of 655 queries ($k = 16$) $n \in 4, 8, 16$. Then I generate as many training and validation sets as the possible distributions of the n
judgments over the four grades (judgment sets). For instance, for \( n = 8 \), one possible distribution would be \((4,1,1,2)\) meaning that each query will have 4 completely irrelevant documents, 1 marginally relevant, 1 relevant and 2 highly/perfectly relevant. Three learning to rank algorithms are run over the thousands of the aforementioned datasets, Rank Boost [42], Linear Regression and Ranking SVM [19]. The performance of each of the algorithms is measured by nDCG at cut-off 10 (nDCG@10). I tuned the algorithms carefully by prior investigation of the effect of feature normalization and on the validation sets corresponding to each of our derived training sets. In the case of SVM, I found prior to our study that feature normalization greatly affects the learning ability of SVM. I took the log of pagerank and similar static document features to put those features on the same scale as text-based (e.g., language model derived) features. Further, I applied zero-mean unit variance normalization in order to make the features scale-free. Rank Boost is not sensitive to feature normalization and I ran it without transforming the features in any way.

To quantify the distribution of labels over the four relevance grades I use two summaries which I found to be particularly informative about the effectiveness of learning to rank algorithms as measured by nDCG@10: (a) the normalized cumulative gain of the judgment set, and (b) the variance over the judgment set. In a further analysis, I summarize the two measures in a single number which is correlated with the overall utility of the label distribution in the dataset.

Figure 4.1 demonstrates the interplay between the distribution of the labels over the different grades in the training sets and the three learning to rank algorithms. In Figure 4.1 (left column), the normalized cumulative gain of the grades of the labeled urls in the training set per query is computed. For instance, for the distribution \((4,1,1,2)\) above with 4 completely irrelevant documents, 1 marginally relevant etc., the cumulative gain is, \(4^0 + 1^1 + 1^2 + 2^3\). The larger the normalized cumulative gain, the more skewed is the distribution of labeled documents towards perfectly relevant documents; the smaller it is, the more it is skewed towards completely irrelevant ones. DCG values in the middle of the x-axis range indicate a
balanced training set with regard to “positive” and “negative” instances. As it can be observed from all three plots on the left, the optimal performance is indeed reached when there is a balance of labels in the dataset. Note that this is not a contradiction to Aslam et al. who report a dropping performance when the ratio of “positive” to “negative” instances exceeds some threshold. In that work and due to the manner the datasets were constructed, one can only view the right tail of what appears a bell-shape curve in the left-hand side plots of Figure 4.1.

Obviously, a number of different datasets can be constructed that have a level of balance in the labels. For instance, a uniform distribution is similarly balanced to an equal distribution of labels in the two middle grades (marginally relevant and relevant) or an equal distribution of labels in the two extreme grades (completely irrelevant and perfectly relevant). To distinguish these cases of balanced datasets we also compute the variance of the grades in the datasets. For instance, for the example distribution (4,1,1,2) above, the variance would be the variance of the flattened list of relevance grades: 

\[[0,0,0,0], [1], [2], [3,3]\].

The higher the variance, the farther the labels are from the middle grades. Conversely, the smaller the variance, the more uniform the distribution. The effectiveness of learning algorithms as a function of variance is shown in Figure 4.1 (right column). As the figure demonstrates, the higher the variance, the more effective the algorithms are. Hence, optimal performance is reached when instances are mostly taken from the extreme grades of relevance.

### 4.2 Effect of the Distribution of Feature Values on the Quality of the Training Set

In prior work by Aslam et al.[8], the textual diversity among relevant and non-relevant documents in the training set was linked to better performance of learning to rank algorithms. In that work, textual diversity was first measured
Figure 4.1: Rank Boost, Ranking SVM, Regression performance (nDCG@10, y-axis) as a function of the normalized cumulative gain of the training dataset per query (x-axis, left) and the variance of the labels in the training dataset per query (x-axis, right).
for every query and then averaged. The diversity for each query was computed as the average of symmetrized KL-divergence between every pair of relevant and non-relevant documents for the query. One limitation of this measure is that it applies to textual features only, and not to learning to rank features. It would be useful to generalize the result of Aslam et al. to arbitrary types of features. One open question in this regard is how to measure how the distance between the relevant and non-relevant documents affects the training set quality as the measure that Aslam et al. has proposed would no longer hold.

Judging by the work of Aslam et al., it should be the case that large distance between the distributions of relevant and non-relevant documents is correlated positively with higher NDCG after training on the corresponding training set. My reasoning, however, suggest that this finding is only partly valid. I hypothesize that there is a threshold after which larger distances between the distributions would lead to performance. This is not a discrepancy between [8] and my work as in Aslam et al. [8] only a part of the range of possible distances between the datasets was observed. The hypothesis that I test in this section is that there are two factors related to the distances between distributions that affect the ability of algorithms to learn: distance between the means of the distributions, and their overlap as determined by their covariance matrices. The overlap could be caused by noise in the labels and by lack of appropriate features to separate the classes.

In order to visualize the MSLR-WEB30K data, I created a dataset with two aggregated features that is based on the original MSLR-WEB30K dataset. One of the features was based on static document features such as spam score and page rank score. The other feature was based on textual features extracted from the body, urls and anchor text. Each of the features had a reasonable correlation with the relevance score (above 0.5) and both features were negatively correlated (around −0.16). For each relevance category from 4 to 0, I produced a plot of the distribution of the documents in that category in comparison with the global distribution of documents from all categories. The plots per category are shown in Figure 4.2. As can be seen from the plots, the distribution of documents in
each category resembles a Gaussian distribution. The distributions of the more relevant documents classes are situated slightly closer to the top right corner of the plot. There is a significant overlap between all distributions. I use the aggregated dataset to make an argument how the distances between the relevant and non-relevant distribution would affect the performance of the learning algorithm. I have abstracted the properties that are visible in 4.2 in an artificial dataset in Figure 4.3. There are two distributions in Figure 4.3. The distribution of the non-relevant documents has a higher variance that causes non-relevant documents to be spread out much further than relevant documents. The distribution of the relevant documents is to the right of the distribution of the relevant documents. However, because the distribution of the non-relevant documents has larger variance there are a number of non-relevant documents to the right of the relevant documents. I imagine three document selection strategies named “High”, “Mid” and “Low”. The “High document” selection strategy chooses documents with high values of feature 1. As one can see from the plot the selected documents from both the relevant and non-relevant distributions are reversed on when project on feature 1. Relevant documents appear to the left of non-relevant documents in dataset “High”. This situation is opposite when the complete distributions are considered. The construction of dataset “High” illustrates that selection of data that is restricted to have feature values above a threshold may fail to work. Since the variance of non-relevant documents is higher than that of relevant documents, it is likely that the non-relevant documents dominate and might reverse the sign of the decision boundary. Revisiting the unsuccessful construction of the LETOR1 collection by Tie et al. [4], we might find some similarities between the artificially constructed dataset “High” and LETOR1. In LETOR1 relevant documents were selected from the complete distribution, but the non-relevant documents were restricted to have high values of BM25. In my experiments in Chapter 5 I evaluated a baseline which selected documents with a feature correlated with the relevance label that had values above a threshold. This baseline was one of the worst performing baselines. As my reasoning here demonstrates the underrepresentation of
non-relevant documents might cause the learning algorithm to fail.

The data selection named “Low” selected both documents that had high and low values of feature 1. As it can be seen the datasets are well separated. Learning algorithms would have an easy time finding a good decision boundary in this case. The problem in this case is that because the documents from both distributions are too well separated the learner would be less correct during testing for documents that fall in between. This reasoning suggests that a dataset called “Mid” which has a smaller gap between the distributions of relevant and non-relevant documents would perform the best.
Figure 4.3: Use of an artificial dataset as an illustration of the effect of distances between relevance distributions on the quality of the dataset. The complete dataset is in the top left plot. Relevant documents are represented by blue squares while non-relevant ones are red crosses. Each of the other plots represents a dataset created by choosing documents from the complete dataset. Dataset “High” chooses documents with high value of feature 1. It results in learners that swap the decision boundary. Dataset “Mid” chooses documents where documents from the non-relevant category are far from the relevant one, while in dataset “Low” non-relevant documents are extremely far.
4.2.1 Experiment design

In this experiment I test the hypothesis that distance between the distributions of relevant and non-relevant documents is predictive of dataset quality. There are two issues that have to be addressed in experimental design. The first issue is how to generate datasets in which the distances between the documents vary considerably. The second issue is how to measure the distances between the distributions of relevant and non-relevant documents.

To solve the issue of creating datasets of varying distances between the distributions, I first rank the documents for each query by a ranking algorithm. In such a ranking, the lower the document is in the ranked list, the greater the distance of this document from the document at the top of the list. For a fixed sorted order of documents, I generated a number of datasets by keeping the set of relevant documents fixed, and varying the selection of non-relevant documents. I took the list of all documents ranked by their score and created two lists while keeping the ranked order: one for the relevant and one for the non-relevant. The relevant documents in the $r$ the dataset were always the top $k$ documents from the list of the relevant ones. The non-relevant were the documents at ranks $r$ through $(r+k)$ from the list of the non-relevant ones. Each query in the constructed collections contained 8 documents: 4 relevant and 4 non-relevant. The described method for generating datasets is illustrated in Figure 4.4.

In the experiments described here, the original dataset was filtered to the set of queries that contain at least four documents from categories 4 to 1 and at least 50 documents from category 0. This setup has the advantage that it allows me to use the same dataset to study the effect between different pairs of categories: 4 vs 0, 3 vs 0, 2 vs 0 and 1 vs 0. While the original collection had multiple grades, the collections in this study had only two relevance grades. Categories 4,3,2, and 1 are treated as relevant while category 0 is always used for non-relevant documents.

The test dataset is always the same: it is the dataset marked as test by the MSLR-WEB30K collection.
Figure 4.4: Illustration of the method for generating datasets to measure the effect of the distance between relevance distributions

### 4.2.2 Analysis

To solve the issue of measuring the distance between the distributions of relevant and non-relevant documents I considered a number of methods: Mahalanobis distance, Bhatacharia distance, and Jensen-Shannon divergence assuming multivariate Gaussian distributions. The Mahalanobis distance is defined as

\[
(\mu_1 - \mu_2)^T \Sigma^{-1} (\mu_1 - \mu_2)
\]

where \(\Sigma\) is the covariance matrix of the complete dataset

\[
\frac{1}{2} \ln \left( \frac{\det \Sigma_{12}}{\sqrt{\det \Sigma_1 \det \Sigma_2}} \right)
\]

where \(\Sigma_{12} = \frac{\Sigma_1 + \Sigma_2}{2}\) is the average of the two covariance matrices.
The Jensen-Shannon divergence between two multivariate Gaussians is closely related to the Bhatacharia distance and both formulas are almost identical.

The first component in Bhatacharia distance and Jensen-Shannon divergence are related to Mahalanobis distance and the Euclidean distance between the means. The second component is related to the ratio of the variances between the two distributions. It is easy to see this relationship in the case of one dimensional Gaussians. In that case \( \det(\Sigma_i) = \sigma_i^2 = \text{var}(x_i) \).

For multivariate Gaussians the determinant of the covariance matrix \( \det(\Sigma_i) \) can be obtained by decorrelating the features through SVD and then multiplying the variances of all dimensions. When a dataset \( i \) becomes more spread out, then \( \det(\Sigma_i) \) will grow.

Do the distances between the distributions explain the observed differences in NDCG@10 due to changes in the datasets? Table 4.1 presents the correlations between Bhatacharia distance and NDCG@10 and Mahalanobis distance and NDCG@10. As explained, some advanced distances between feature vectors take into account the variances of the individual distributions. Therefore, it was important to check the explanatory power of various meaningful combinations of the variances of the distributions. The data used to construct 4.1 was computed over 46 collections (by changing \( r \) from 1 to 46) in each of the four category combinations \( i \) vs 0, for \( i \in 4, 3, 2, 1 \). Five different learning algorithms were run on each collections: LambdaMart, LambdaRank, Ranking SVM, Rank Boost and Linear Regression. Therefore, the correlations were computed over \( 920 = 46 \times 4 \times 5 \) data points, where each point is a tuple of the measurement of a collection property and NDCG@10. The results show that the Mahalanobis distance, the simpler of the two distances used, was better correlated with NDCG@10. However, individual components from Bhatacharia distance that are related to the variances of the the distributions of relevant and non-relevant documents show high correlations as well. In particular, the combination \( \ln(\det(\Sigma_{rel})) - 0.5 \times \ln(\det(\Sigma_{nonrel})) \) shows that the better datasets had large variance in the distribution of the relevant documents, while small variance in the distribution of non-relevant documents. This
Chapter 4. Empirical Studies of the Properties of Datasets for Learning to Rank

### Table 4.1: Correlation between the distance between the distributions of relevant and non-relevant documents and NDCG@10

<table>
<thead>
<tr>
<th>Method</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mahalanobis distance</td>
<td>0.54</td>
</tr>
<tr>
<td>Bhatacharia distance</td>
<td>0.49</td>
</tr>
<tr>
<td>$\ln (\det(\Sigma_{rel+nonrel}))$</td>
<td>0.11</td>
</tr>
<tr>
<td>$\ln (\det(\Sigma_{nonrel}))$</td>
<td>-0.57</td>
</tr>
<tr>
<td>$\ln (\det(\Sigma_{rel}))$</td>
<td>0.34</td>
</tr>
<tr>
<td>$\ln (\det(\Sigma_{rel})) - 0.5 \times \ln (\det(\Sigma_{nonrel}))$</td>
<td>0.61</td>
</tr>
<tr>
<td>$\ln (\det(\Sigma_{rel+nonrel})) - 0.5 \times \ln (\det(\Sigma_{nonrel}))$</td>
<td>0.41</td>
</tr>
<tr>
<td>Regression on components in Bhatacharia distance</td>
<td>0.68</td>
</tr>
</tbody>
</table>

Finding can be explained in the following way. When the non-relevant documents are spread out, they may mask the distribution of the relevant documents.

In Figure 4.6 I present plots of NDCG@10 and Mahalanobis distances for five ranking methods: LambdaMart, LambdaRank, Ranking SVM, Rank Boost and Linear Regression. The figure shows the relationship between NDCG@10 and the Mahalanobis distance of relevant documents from category $i$ and non-relevant ones (category 0). The relevance category was varied and had values 4, 3, 2 and 1. In the plots in Figure 4.6, each point represents a dataset. The x-axis measures the Mahalanobis distance between the distributions while the y-axis gives the NDCG@10 for the dataset.

As can be seen from the plots, as the distance between the relevant and non-relevant documents increases, the performance improves sharply. After a particular threshold the performance of the algorithms levels off or degrades slightly.

### 4.3 Interaction between Feature Values and Relevance Categories

The experiment from Figure 4.6 suggests that on average the datasets with relevance category 1 performed better than the datasets with relevance category 4. Since relevance is one of the most important dimensions in the dataset for learning
Figure 4.5: Effect of distance between relevance categories on dataset quality measured by NDCG@10. Each data point represents a dataset. The points of the same color represent the performance of the dataset for the same algorithm. One can see that higher values of the distance between distributions correspond to higher NDCG@10.
to rank, the results from this section may seem counter intuitive. In fact the experiment on Figure 4.1 shows that datasets that have documents from category with a higher relevance grade tend to perform better. How can this two seemingly contradictory findings be reconciled? My hypothesis is that both the (1) relevance grade and the (2) diversity of documents in the constructed datasets are important. Both studies (Figure 4.6 and 4.1) were designed to measure individually each of those aspects. In the study of the distances between relevance categories the diversity aspect dominates. Therefore, I expect that the documents from category 1 are more diverse than the documents from category 4. In order to substantiate this claim, I computed one measure of diversity of a dataset: the determinant of the covariance matrix of the documents from category $i$, $(i \in \{4, 3, 2, 1\})$. This measure is suggested by the theory of Optimal Design of Experiments presented in Chapter 3. The results are shown in Table 4.2 and confirm that higher values of the log of the determinant of the covariance matrix correlate with higher NDCG. This finding is useful because it suggests why there is significant value in documents from categories of marginal relevance.

Table 4.2: This table provides an explanation why less relevant categories might still be useful. The less relevant categories turn out to contain larger feature variability which is positively correlated with better performance.
Figure 4.6: Effect of distance between relevance categories on performance measured by NDCG@10. Each data point represents a dataset. The points of same color represent the performance of the dataset for the same algorithm. One can see that higher values of the distance between distributions correspond to higher NDCG@10.
4.4 Effect of Number of Documents per Query

When designing a learning to rank dataset one can opt for a large number of queries to obtain a good coverage of all possible queries. Alternatively, one can try to select a smaller number of queries each of which has more documents per query. Which strategy is better? An experiment by Yilmaz et al. [10] shows that the strategy with more queries is better in the case of the LambdaRank learning to rank algorithm. At the end of their paper, Yilmaz et al. explain that adding a new query to the training set brings more information to the training set rather than adding new documents to an existing query. A new query is likely to emphasize new features or feature combinations and so it brings diversity in the training set.

In this section I verify whether the conclusions of Yilmaz et al. hold both when (1) different learning methods are used and when (2) different collection construction algorithms are used.

4.4.1 Experiment “Variable Number of Documents per Query”

In this experiment the number of documents per query varies from 6 to 45, but the total number of documents equals one of three possibilities: 8000 total documents, 16000 total documents, and 64000 total documents. As I needed to measure the effect when the number of documents per query is rather large, I pre-selected queries with more than 80 documents per query from the LETOR dataset. The number of the selected queries was 16694. The selected queries constituted about 72 percent of all queries in the dataset. Using these queries, I generated a number of datasets having 6, 8, 10, 12, up to 45 number of documents per query. The total number of documents in each of the datasets was the same. The results for five learning to rank algorithms over three possible total number of documents are shown in Figure 4.8. I repeated the experiment twice to reduce the variability in the results.
As it can be seen from the Figure 4.8, the LambdaMart and Rank Boost were most sensitive to the number of documents per query. The sensitivity was most visible (1) when the dataset had fewer total documents and (2) for non-linear model. Both LambdaMart and Rank Boost are algorithms which combine features in a non-linear way. The effect was largest for those algorithms. In Figure 4.8 I averaged the NDCG@10 over all learning models and dataset sizes. In this figure, one can clearly see a trend that less documents per query (and more queries) is better. I also plotted the variance of NDCG@10 for each value of the number of documents per query. The variance is computed over 10 data points: 5 algorithms, each repeated on two datasets. The plot shows that the variance of NDCG@10 due to the data selection strategy increases as the number of queries decreases. This result confirms the finding of Yilmaz et al. It also shows that non-linear (higher variance) models are more sensitive to the number of documents per query than models using linear feature combinations.

### 4.4.2 Experiment “Number of Documents per Query for Different Selection Strategies”

Since Yilmaz et. al. only carried out their experiment with one random document selection strategy, one might ask if the same property would hold if a different selection strategy was used.

I experimented with a document selection strategy which combines random and relevant documents (“random + relevant” selection). In this experiment I studied the effect of number of documents per query (denoted by \(k\)) over a large number of different collection sizes (denoted by \(D\)). I used the values \(k \in \{4, 8, 16, 32\}\) as the number of documents per query. I evaluated those four types of datasets over multiple training collection sizes \(D\) starting from 136 total number of documents per query to 50000 (\(D \in \{136, 272, \ldots, 1360, 2000, 3000, 4000, \ldots 48000, 49000, 50000\}\)). The datasets created for each value of \((k, D)\) were created by picking a random set of queries. Once a query is selected, I picked \(k/2\) documents per query that were on top of the ranked list produced by a previously trained ranking model and
Figure 4.7: Effect of the number of documents per query over dataset quality. In each of the graphs the points represent datasets with the same total number of documents in the collection, but different number of documents per query.
Figure 4.8: Effect of the number of documents per query over NDCG@10. In this plot the NDCG@10 from Figure is first aggregated for each value of the number of documents per query. The left plot shows the average NDCG@10 over all learning algorithms, while the right plot show the variance of NDCG@10. As it can be seen, the datasets with less queries have higher variance and lower NDCG@10.

$k/2$ additional documents randomly selected from the rest. This experiment confirmed showed clearly that fewer documents per query works better even for the “random + relevant” document selection strategy. The results appear in Figures 4.9 and 4.10. Figure 4.9 shows the results when $k$ is one of 4,8,16 and 32. Figure 4.10 compared the results for the two most extreme values of $k = 8$ and $k = 32$ to make the effect stand-out. In this case, the influence of $k$ can be clearly seen. Smaller values of number of documents per query ($k$) performed better.

### 4.5 Conclusion

In this chapter, I conducted exploratory data analysis of the dataset properties as they relate to learning ranking functions in Information Retrieval. The experiments have shown that diversity among relevance grades and features are useful properties of good learning to rank datasets. The experiments delivered some surprising results such as the finding that less relevant documents are still useful and that fewer number of documents per query, but more queries are more beneficial for non-linear models.
Figure 4.9: Effect of the number of documents per query over NDCG@10 for training collections of varying sizes. The datasets here are a combination of random and top-ranked documents.
Figure 4.10: Effect of two extreme values of the number of documents per query over NDCG@10 for training collections of varying sizes. This figure is based on Figure 4.9. The number of documents per query are restricted to two extreme values: 8 and 32. The datasets were generated by a combination of random and top-ranked documents.
5 Application of the Theory of Optimal Design of Experiments to Construction of Training Collection for Learning to Rank

5.1 Introduction

In this chapter I evaluate multiple document selection strategies over four learning to rank algorithms. Some of the evaluated document selection strategies are known from the literature and some are new. The new data selection strategies are motivated by the theory of the Optimal Design of Experiments (ODE) introduced in Chapter 4. The intuition behind ODE for linear regression is that the selected documents are maximally spread apart. ODE can be applied to classification, and in the case of logistic regression, the optimal documents are simultaneously spread apart and are situated around the decision boundary. Intuitively, similar properties of the dataset should also hold for ranking tasks. Following the analysis of ranking in this chapter, the most informative documents among the documents ranked by a previous learning model are the top-ranked documents. At the same time, using the intuition of ODE, documents should be spread apart. The data selection strategies evaluated in this chapter illustrate the interplay between these two properties across a number of learning algorithms. The mentioned properties of the data selection strategies are reinforced by the empirical findings in Chapter 4. According to the results in Chapter 4 the following factors are important for the quality of the training set: (a) the ratio between relevant and non-relevant documents and (b) the distances between documents of different relevance grades. The top-ranked documents can serve as a proxy for the relevant documents. The documents selected by design to be far from the top documents can be viewed
as an approximation to the distances between documents from different relevance categories. Therefore, this chapter can be considered as a practical application of the properties of the training collections, established in Chapter 4.

The evaluation of document selection strategies is carried over four learning to rank algorithms. The learning algorithms I use in this study are among the most widely used in the IR literature. Learning algorithms, as suggested in Chapter 2, can be classified by two important dimensions: (a) the error function and (b) the method of expressing feature combinations (e.g. linear vs. non-linear feature combinations).

The main hypothesis in this chapter is that the optimal dataset for each learning to rank algorithm is one that mixes a number of top-ranked and diverse documents. The ratio between those two groups of documents depends on the properties of the learning algorithm and the dataset size. For example, algorithms with linear feature combinations are more sensitive to top-ranked documents, while non-linear algorithms show better performance when documents are spread apart. Top documents have a greater positive effect when the dataset size is large. In datasets with more documents, the optimal distances between top and other (non-top) documents are smaller.

5.2 Related Work

The study of Aslam et al. [8] has shown that the (1) ratio of relevant to non-relevant documents in the learning to rank training datasets and (2) the distances between relevant and non-relevant documents are two important factors for the quality of the training dataset. Their study, however, does not show how those factors depend on the learning to rank algorithm. My findings show that the optimal combination of those two factors depends on the learning algorithm. Therefore, it is possible to have a data selection method on which learning algorithm A could perform extremely well, but learning algorithm B would fail. The study of Aslam
et al. has not shown data selection strategies that combine explicitly the above mentioned factors and has not attempted to improve upon the performance of well-known data selection methods. In contrast, one of my goals in this study is to improve upon the baseline methods.

The study of Aslam et al. was carried out over datasets with very few features. When their study was carried out there were no large suitable collections for learning to rank. In their study the learning algorithms were able to learn well using very small datasets, so the effect of the data selection strategies was observed only over very small datasets. This makes one wonder whether the same conclusions would hold if the study was carried out over datasets with more features and more documents. In my study I improve upon these two dimensions. The experiments that I present in this chapter are carried out over the two largest learning to rank datasets available to IR researchers. The datasets I use have a large number of features (100 - 600) and a large number of documents (up to 150000).

5.3 Experimental Design

5.3.1 Datasets

This study is based on the two largest learning to rank datasets available to researchers: Microsoft MSLR-WEB30K datasets \(^1\) and the SET 1 dataset from the Yahoo! Learning to rank challenge [7] referred to here as the Yahoo! LTR dataset. The Microsoft MSLR-WEB30K collection consists of more than 30000 queries that were once used to train the Microsoft Bing search engine. There are 136 document features per document. The features are well known in the Information Retrieval community. One notable omission from the list of features are word proximity-based features. While little is known how this dataset is constructed, it is reassuring that the documents (but not the features that come with MSLR-WEB30K) have been used to train a production system. Therefore,

\(^1\)http://research.microsoft.com/en-us/projects/mslr/
one could expect that this collection is of reasonable quality. Some properties of
the MSLR-WEB30K collection are presented in Table 1.1.
The MSLR-WEB30K dataset has five folds each of which has a separate training,
testing and validation sets. I have chosen to build my own folds using fold1. In this
way, I change only the sets from which various learning datasets are constructed
as training sets. I use the test set of fold 1 for evaluation of all the data selection
strategies. I use the validation set of fold 1 MSLR-WEB30K to obtain a regression
model that assigns a ranking score to each document from the training set. I split
the training set in 6 folds, so that each fold contains a random selection of 75%
of the documents of the training set. I run the document selection procedures on
each of the six folds obtained from the training set. For each of the six folds and
each data selection method I generate datasets of various sizes. Each generated
set is spit into two parts: a training set and a validation set for use by the learning
to rank algorithm. The processing steps are summarized in the list below:

- Fix fold1 test set of MSLR-WEB30 as a test set to be used by all data
  selection methods and learning algorithms.

- Use fold1 validation set to obtain a linear regression ranking model. The
  model is used to assign relevance scores to each document from fold1 training
  set.

- Take six random samples of fold1 training set, where each sample selects
  without replacement 80 % of the queries of the fold1 training set.

- Apply each data selection method to each of the six folds generated at the
  previous step to produce a collection of documents of given size while using
  a particular methodology.

- Split each training collection from the previous step into two parts: a training
  part (80%) and a validation part (the remaining 20%).

- Apply each learning algorithm to the training-validation pair and learn a
  ranking model using each of the four chosen learning to rank algorithms.
Figure 5.1: Distribution of the number of documents per query. The left plot shows the distribution of documents from all relevance categories. The right plot shows the distribution of highly relevant documents. As it can be seen, most queries in MSLR-WEB30K have a sufficient number of documents, but most queries do not have highly relevant documents.

- Run the learned ranking model on the complete test set of fold1 of MSLR-WEB30K.

A graphical illustration of the methodology appears in Figure 5.2.

The Yahoo! LTR dataset was created to give a chance to researchers to develop learning to rank algorithms on state-of-the-art training collections. The dataset was released as part of the Yahoo! learning to rank competition in which multiple research groups competed to produce the best ranking function. The Yahoo! dataset was based on data from the Yahoo! search engine and state-of-the-art proprietary features. Some statistics of the Yahoo! dataset are given in Table 1.1.

The Yahoo! LTR dataset has some notable differences from the MSLR-WEB30 dataset. First, the test set that comes from the Yahoo! learning to rank competition is not available for obvious reasons to the public. Therefore, I needed to produce my own training, validation and test sets using the training set provided by Yahoo!. In this way, I was able to use the same experimental setup as for
Chapter 5. Application of the Theory of Optimal Design of Experiments to Construction of Training Collection for Learning to Rank

the MSLR-WEB30 set. A notable difference between the Yahoo! and the MSLR-WEB30 training set is the way the sets were created. The Yahoo! LTR dataset has been created especially for the purpose of the Yahoo! competition by depth pooling using a number of search engines. Another difference between MSLR-WEB30 and Yahoo! LTR dataset is the number and types of features used. MSLR-WEB30 used features known in the research community while the Yahoo! dataset has proprietary features used by the Yahoo! search engine. The Yahoo! dataset used an inverse cumulative transformation of the features. I will illustrate this transformation for a feature with four values -0.2, 0.12, 0.15 and 0.8. Those values would be transformed to 0, 1/3, 2/3, and 1 by the inverse cumulative transformation. In contrast, MSLR-WEB30 does not use such a transformation, but instead uses the values of the raw features. Yet another difference between MSLR-WEB30 and Yahoo! is the number of documents per query. While the average number of documents per query for the MSLR-WEB30 datasets is 120, the average for Yahoo! LTR dataset is only 24. As it can be noticed, the Yahoo! dataset has much fewer number of documents per query. It can also be observed that the percentage of non-relevant documents (category 0) in the MSLR-WEB30K dataset is much higher than in the Yahoo! dataset. For the purpose of the study of training collections, queries with larger number of documents are preferable. This is so because with more documents per query, the document selection procedure would have a greater opportunity to perform well or badly, and therefore the observed effect due to document selection is likely to be larger. In my experiments I filtered out queries with less than 20 documents. This left 5836 queries or about 50% of all queries.

5.3.2 Learning algorithms

The performance of a data selection method can only be evaluated indirectly, when a number of learning algorithms are applied to the training set generated by a particular method. I selected four learning to rank algorithms often used in the research literature. Those algorithms are linear regression, Ranking SVM, Rank
Figure 5.2: The figure illustrates the experimental design for testing a document selection method. A number of document selection methods operate on the training dataset and generate smaller datasets of varying sizes. Those datasets are used for training and evaluated on the same test dataset. The final results are presented in a plot. Each point in the plot represents the performance of a dataset of some size originating from some data selection strategy.
Chapter 5. Application of the Theory of Optimal Design of Experiments to Construction of Training Collection for Learning to Rank

<table>
<thead>
<tr>
<th>Learning Algorithm</th>
<th>Error Criterion</th>
<th>Feature Combinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>LambdaMart</td>
<td>NDCG approximation</td>
<td>Non-linear (trees)</td>
</tr>
<tr>
<td>Rank Boost</td>
<td>Pairwise error</td>
<td>Non-linear (stubs)</td>
</tr>
<tr>
<td>Ranking SVM</td>
<td>Pairwise error</td>
<td>Linear</td>
</tr>
<tr>
<td>Linear Regression</td>
<td>Squared error</td>
<td>Linear</td>
</tr>
</tbody>
</table>

Table 5.1: Some characteristics of the learning algorithms

Boost and LambdaMart. Each of those algorithms has different properties than the others. For example, Ranking SVM uses the pairwise error in a query as a training criterion and linear combination of features. LambdaMart uses an error criterion which approximates the NDCG and non-linear combination of features. The properties of these algorithms are given in Table 5.1. The algorithms were described in detail in Chapter 2. Some of the mentioned algorithms have available open source implementations. For Ranking SVM I used the implementation of Joahims\(^2\) and for LambdaMart I used the implementation of Ganjisaffar \(^3\). I used the following configurations to train SVM and LambdaMart:

SVM command: `svm_rank_learn -v 1 -c 50 -l 2 -w 1 -d 10 000 -b 50 -t 0 -b 100 -k 100 -e 0.01 <TRAINFILE> <MODELFILE>`

LambdaMart configuration file:

```
trees.num-leaves=7
trees.min-instance-percentage-per-leaf=0.25
boosting.learning-rate=0.05
boosting.sub-sampling=0.3
trees.feature-sampling=0.3
boosting.num-trees=500
learning.algorithm=LambdaMart-RegressionTree
learning.evaluation-metric=NDCG
params.print-intermediate-valid-measurements=true
```

5.4 Document Selection Procedures

In this study I evaluated a large number of document selection procedures. Document selection procedures can be categorized according to multiple aspects. One dimension is whether the document selection procedure uses an update loop as in active learning or operates in batch mode as is typical for IR collections. All the procedures that I utilized operate in batch mode. Active learning, however

\(^2\)http://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html
\(^3\)http://code.google.com/p/jforests/
interesting, is beyond the scope of this study. There is some evidence suggesting that while active data selection improves the performance of the learning to rank algorithm, the produced dataset may not be useful to other learning to rank algorithms. Document selection procedures in Information Retrieval usually operate in batch mode, so active learning methods are typically not even considered as candidate procedures for collection construction.

Another dimension of the learning datasets is the extent to which they use information about the relevance of the documents. A search query typically has only a few relevant documents while the rest are non-relevant. Intuitively, a document selection procedure that selects only non-relevant documents would be completely useless for learning. To prevent such a situation from happening the document selection procedure needs to make sure that enough relevant documents are selected. I consider two types of document selection procedures: (1) those that explicitly add top-ranked documents since they are most likely to be relevant and (2) those that do not explicitly add top documents. In my experiments, each dataset selection method uses either directly or indirectly some relevance information. The indirect use of relevance information is due to the fact that the documents are selected from a previously constructed training collection (MSLR-WEB30, Yahoo! LTR dataset) where the relevance aspect has been incorporated into the data selection. Those collections were constructed in such a way that they are likely to contain relevant documents. The data selection methods that use directly information about the relevance of documents obtain this information through the use of a relevance score. The relevance score is obtained from a learning model (linear regression) trained on a different dataset and serves as a proxy of the actual relevance label.

All of the data selection algorithms tested here use a common framework for the document selection while they differ in a few specific aspects. A common feature is that each procedure selects a random set of queries and applies a data
selection procedure to the documents within each query. It is known from the literature (see e.g. [10]) that training sets with more queries (and less documents per query) are more informative than training sets with fewer queries. Therefore, using as many queries as possible is likely to give an advantage to each data selection strategy. Thus the data selection strategies would be compared at settings which are likely to make them most effective.

Based on prior work (e.g. [10]) and some initial experiments I found out that training collections with fewer documents per query and higher ratio of top-ranked to other documents perform better. Therefore, I fixed the number of documents that are selected per query to six. However, when a data selection procedure needed to create a larger dataset, there may not have existed enough queries in the MSLR-WEB30K or Yahoo! LTR datasets to create a dataset with the required number of total documents. In that case, more documents need to be selected per query. Because of this reason, each data selection procedure I experimented with is based on the following framework:

- Draw a random query with replacement.
- If the query has not already been selected, then draw six documents according to the data selection procedure.
- If the query has already been selected, then draw 3 extra documents from the remaining unused documents.

In what follows, I would describe in detail each of the data selection methods employed. For convenience all the data selection methods are summarized in Table 5.2.

### 5.4.1 Selection of random documents (“Random”)

As the name of this procedure suggests it applies random selection. For a given query, six documents are selected without replacement from the pool of
## Table 5.2: Parameters of the data selection methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Num. top docs.</th>
<th>Num. other docs.</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Top by score</td>
<td>6</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Random + top by score</td>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Depth</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>StatAP</td>
<td>0</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>ODE</td>
<td>0</td>
<td>6</td>
<td>Feature transformation (see text)</td>
</tr>
<tr>
<td>ODE + top by score</td>
<td>4</td>
<td>2</td>
<td>Feature transformation (see text)</td>
</tr>
<tr>
<td>Random Projection</td>
<td>4</td>
<td>2</td>
<td>Feature transformation (see text); Uses a threshold parameter</td>
</tr>
<tr>
<td>Random + top by BM25</td>
<td>0</td>
<td>6</td>
<td>Uses BM25 score instead of regression score</td>
</tr>
<tr>
<td>Top by BM25</td>
<td>0</td>
<td>6</td>
<td>Uses BM25 score instead of regression score</td>
</tr>
</tbody>
</table>
available documents if this query has not been selected before. Otherwise, three extra random documents are added.

5.4.2 Selection of top-ranked documents (“Top by score”)

Under this procedure, the documents for a query are first ranked by a ranking score. The ranking score is obtained from a linear regression model as described earlier. The top six documents by ranking score are selected. If a query has already been selected, then three extra top documents are added.

5.4.3 A mixture of random documents and top-ranked documents (“Random + top by score”)

This procedure first ranks the documents for a query by the ranking score and then selects the top four documents. Then another two documents are selected randomly without replacement from the remaining documents. If a query has already been selected, but more documents from the query are needed, I select two top documents and one random. Therefore for any query, the ratio between top documents and randomly selected documents is kept as 2:1.

5.4.4 Selection based on depth pooling (“Depth pool”)

Depth pooling requires access to a number of rankers which would produce a set of ranked lists. In this case, I have chosen each feature to work as a weak ranker. This procedure, similarly to the other procedures, is constrained to use six documents per query initially. Therefore, the depth pooling procedure operates as follows. For each query depth pooling is applied for each depth starting from 1, 2, 3, and so on, until enough documents per query are selected. If all 136 features in the MSLR-WEB30K collections had a different document at depth 1, then out of those 136 unique documents, I would select six randomly. If all features ranked
the same document as the top one, I would continue the same procedure using depth two and so on.

5.4.5 Selection based on StatAP ("StatAP")

The implementation of statAP is based on the description in [1]. Similarly to depth pooling each feature is used as a weak ranker. The ranking order of a feature implies a sampling distribution for the feature as specified in [1]. The sampling distributions for all features is averaged. The average sampling distribution is used to split the documents into buckets of six documents. All the documents in the first bucket are selected. In order to select more documents per query, when there is a need, the statAP procedure is executed again with bucket size of 9 documents, 12 documents, and so on.

5.4.6 Selection of documents with high BM25 score ("top by BM25")

Under this procedure, the documents are first sorted by BM25 score. The six documents with the highest BM25 score were selected. This procedure is similar to the LETOR procedure tested in [8].

5.4.7 Selection of random documents and documents with high BM25 score ("Random + top by BM25")

Under this procedure, the documents are first sorted by BM25 score. The four documents with highest BM25 score are selected. A random selection of two extra documents is made among the remaining documents. This procedure is similar to the document selection procedure “A mixture of Random and Top-ranked Documents”, with the only difference being that the documents are sorted by BM25 score rather than regression score.
5.4.8 Selection of documents based on ODE without top documents ("ODE")

Under this procedure, the queries are chosen consecutively by random sampling with replacement. When a new query is chosen, six documents for this query are selected based on the greedy approximation algorithm from Chapter 4. The information matrix is updated after a new document is chosen and is used for the queries selected subsequently.

The application of this document selection method is sensitive to feature preprocessing. For the MSLR-WEB30 dataset, the following transformations are applied.

- Rank the documents for each query by relevance score
- Remove bottom-ranked documents whose score is less than a threshold from the score of the top-ranked document. Here the threshold was chosen as 1.0
- Apply the inverse distribution transformation to each feature. This transformation is described above, in the section of the Yahoo! LTR collection.
- Normalize each feature by subtracting the mean and dividing by the standard deviation

For the Yahoo! LTR dataset, I carried out only the last step since the previous steps have been carried out during the design of the collection.

5.4.9 Selection of documents based on ODE with top-ranked documents ("ODE + top")

Similarly to the previously described data selection method based on ODE, this dataset selection procedure sampled queries randomly. Once a query was chosen, the documents were ranked by their relevance scores. The top four documents were added to the dataset. The remaining two documents were selected based
Chapter 5. Application of the Theory of Optimal Design of Experiments to Construction of Training Collection for Learning to Rank

on the greedy algorithm from Chapter 3. This method ensured the selection of top-ranked and diverse documents.

5.4.10 Selection of documents based on projection of documents on a random line (“Random projection”)

Application of the theory of ODE to ranking suggests a transformation in feature space which would emphasize documents with high ranking scores. The suggested transformation is multiplication of the document feature vector by the gradient as computed by methods such as RankNet. In Chapter 3, it was made clear that all of the pairwise learning algorithms improve by reducing the average pairwise error. The pairwise error is the number of incorrectly ordered pairs in which a given document participates. The gradient for RankNet is related to the pairwise error (Formula 2.20 in Chapter 2). A plot of the gradient of RankNet by the rank of the document in a ranked list ordered by relevance scores is shown in Figure 5.3. The procedure to obtain this plot is as follows. First, a linear regression model was trained on a separate dataset. The linear model was used to assign scores to the documents from a training dataset. The documents for each query were ranked by decreasing score. The Formula 2.20 from Chapter 2 was used to obtain a gradient for each document. The multiple queries in the collection were reduced to list of tuples (document rank, document gradient). This list was grouped by document rank, and the average gradient for each rank was computed.

The above plot shows that the top documents would have the largest effect on the learning procedure.

In what follows, I present a randomized data selection algorithm which incorporates the properties of the data collection suggested by ODE. The randomized algorithm is presented in Figure 2. The algorithm incorporates a number of key ideas from ODE. The first idea is that the data selection for the linear model can be mimicked by projecting all points on a random line and then choosing the furthest
two points. Provided enough random projections are made, the boundary of the ellipsoid containing the data set would be covered with points. The second idea is that documents with highest relevance scores are very important. This property is justified by Figure 5.3. Therefore, given a query, the selection of the document with the highest relevance score is an obvious choice. The selection of another document for the query is constrained to the subset of documents which score is within a threshold of the top document score. There are two factors that inform this property. First, as suggested by Figure 5.3, the most informative documents have high ranking scores. However, if the scores are too close it is mostly likely that a learning algorithm would not be able to tell those documents apart because of the noise. Therefore the noise caused by the overlap between the relevant and the non-relevant distribution forces at least a minimum threshold between the scores. Reasoning in this was, we can deduce that neither too low a threshold nor too high a threshold is optimal. The optimal threshold would depend on the collection size and the overlap between the distributions of relevant and non-relevant documents. This procedure was applied with six top documents per query and three random
Chapter 5. Application of the Theory of Optimal Design of Experiments to Construction of Training Collection for Learning to Rank

documents. For the selection of a new random document, a new random line was
drawn, and the document furthest from the topmost document for the query was
selected. Zero mean and unit variance normalization was applied to each feature
in the collection.

Algorithm 2 Random Projection: A Randomized Algorithm for Collection Con-
struction based on ODE

Use a previously available model to assign relevance score to each document

while More data points required do
    Choose a random query $q$
    Filter out documents from $q$ already included in the collection
    Choose a random line through the origin $L$
    Find the document in query $q$ with largest relevance score and call it $topDoc$
    Remove all documents $d$ with scores $topDoc.score - d.score > threshold$
    Project all remaining documents onto the random line $L$
    Choose the document $farthestDoc$ which has projection on $L$ farthest from $topDoc$
    Add documents $topDoc$ and $farthestDoc$ to the collection
end while

5.5 Results and Analysis

In this section I analyze the performance of the proposed data selection meth-
ods (ODE and Random Projection) and compare them with the previously avail-
able data selection methods (depth pooling, statAP, top, etc.). The evaluation of
the performance of a dataset can be done indirectly through the use of multiple
learning algorithms. If a learning algorithm learns well on a given dataset, then
for the purpose of this study, the success is attributed to the dataset, since the
datasets are the variables that change rather than the algorithms. For each data
selection method a number of datasets of different sizes were generated. This ex-
perimental design has the advantage that one can observe how the performance
depends on the size of the data. Additionally, it provides multiple data points
by which to judge the data selection methodology. If the resulting line on the
plot wiggles too much, then the data selection method is unstable. The results
Figure 5.4: Comparison between methods that do not use top-ranked documents on MSLR-WEB30K collection
Figure 5.5: Comparison between methods that do not use top-ranked documents on Yahoo! LTR collection
Figure 5.6: Comparison between the methods depth pooling and statAP on MSLR-WEB30K collection
Figure 5.7: Comparison between the methods depth pooling and statAP on Yahoo! LTR collection
for each combination of data selection method, learning algorithm and underlying dataset (MSLR-WEB30K and Yahoo! LTR dataset) are plotted in two dimensions. The x-axis represents the sizes of the evaluated datasets. The y-axis represents NDCG@10. Each data point represents the performance of a dataset of a particular size. Each data point was obtained by averaging runs over six folds. The vertical bars around each point are one standard deviation away from the mean.

The simultaneous presentation of 12 document selection methods (including those that differ by a single parameter) is likely to be awkward. Therefore I present groups of a few related methods together. In that way, methods can be easily compared.

The presentation of the results differentiates between methods that explicitly use top-ranked documents and methods that do not use such documents.

5.5.1 Methods that do not select explicitly top-ranked documents

Among this group of methods are random selection, depth pooling, statAP and ODE. The methods random selection, depth pooling and statAP are known from the literature, while ODE is a new method. The performance is shown in Figure 5.4 for the Microsoft MSLR-WEB30 collection and Figure 5.5 for the Yahoo! LTR dataset. As it can be seen, the ODE method has a much larger effect on the MSLR-WEB30 collection than on the Yahoo! LTR collection. The ODE method outperforms the baselines for all learning to rank algorithms. For the Yahoo! LTR dataset, the best performing method is much less clear. For this dataset, all the data selection methods are much closer in performance, but depth pooling and ODE perform better than random selection. For the learning method LambdaMart, ODE outperforms depth pooling. The difference in performance between the MSLR-WEB30K and the Yahoo! dataset is most likely due to the number of documents per query in each dataset. Since the Yahoo! LTR dataset, has much fewer documents per query, the difference between any method and
random selection is likely to be less than in the case of the MSLR-WEB30K dataset. In Figure 5.6 and 5.7 I present the results for depth pooling and statAP. As it can be seen both methods are rather close with statAP having slightly lower performance.

5.5.2 Methods that explicitly select top-ranked documents

The methods that explicitly select top-ranked documents are top, random + top, ODE + top and random projection. First, I evaluate the baselines: top vs. random vs. random + top. The results are shown in Figures 5.8 and 5.9. One immediate observation is that for all learning algorithms except for LambdaMart, the method “top” performs better than “random”. In LambdaMart, a selection of top documents is less effective than random selection. A combination of random and top documents is always better than random selection but not always better than “top”.

The comparison between random projection and random + top is shown in Figures 5.10 and 5.11.

5.5.3 Use of the BM25 score instead of the relevance score

This experiment illustrates the value of ordering the documents by regression score, rather than a single feature like BM25. Figure 5.12 shows a comparison between two document selection strategies (top and random + top) using two scores. The two scores are BM25 and the regression score. As it can be seen, using a feature score like BM25 instead of a regression score leads to decrease of the quality of the training set.

Since the feature names are only available for the MSLR-WEB30K collection, this experiment is only possible for the MSLR-WEB30 collection.
5.5.4 Interaction between the data selection methods, learning algorithms and dataset sizes

The analysis of the performance of the many data selection methods presented above reveals that certain data selection methods performed better for certain learning algorithms. For example, the data selection methods that combine top-ranked documents with random documents performed much better for Ranking SVM, while depth pooling performed better for LambdaMart. Can one relate the performance of the data selection methods to the properties of the learning algorithm?

The hypothesis that I suggested in the beginning of the chapter is tested here. Each learning algorithm can be represented by two properties: (1) the error function and the (2) feature combination. These properties are presented in Table 5.1 for each of the four learning to rank algorithms used here. Some of the selection method can be represented as a combination of top and diverse documents of various proportions. The diverse documents are obtained either through random choice or through the use of distances as in ODE and Random Projection. If all the data selection methods are plotted simultaneously, as in Figure 5.13 and Figure 5.14, it can be observed that the NDCG@10 curves cluster into two groups. For Ranking SVM, the top performing datasets use a combination of top and random documents. The comparison between the methods Random projection (threshold = 0.5) and Random projection (threshold = 1) shows that the dataset with the smaller distance between the top and other documents performs better for Ranking SVM. LambdaMart, demonstrates the opposite effect: it performs better on the dataset with larger distances. It can also be observed that ODE (without top documents) performs better than ODE with top documents for LambdaMart, but exactly the opposite holds for Ranking SVM. It can also be observed that depth pooling outperforms random + top in LambdaMart, but random + top outperforms depth pooling for Ranking SVM.

One could also observe that for different sizes of the training dataset, the best performing data selection methods change. For example, for smaller dataset sizes,
as seen in Figure 5.10 and Figure 5.11, the random projection method has different optimal values of the threshold. For smaller datasets, larger threshold is better. The larger threshold corresponds to larger distances between the documents.

Altogether, my experiments show that there is no universal dataset which is optimal for all learning to rank algorithms. This is reminiscent of the no free lunch theorem for machine learning, which states that no learning algorithm would outperform all other learning algorithms on all problems. One practical conclusion of my experiments is that any collection constructed without properties of a particular learning algorithm in mind is likely to provide sub-optimal training data. In spite of this negative finding, my experiments also showed a class of data construction methods which create training collection of reasonable quality for multiple learning to rank algorithms.
Figure 5.8: Comparison between methods that use top-ranked and random documents on the MSLR-WEB30K collection
Figure 5.9: Comparison between methods that use top-ranked and random documents on the Yahoo! collection.
Figure 5.10: Comparison between methods that use top-ranked documents on the MSLR-WEB30K collection
Figure 5.11: Comparison between methods that use top-ranked documents on the Yahoo! collection
Figure 5.12: Effect of selecting top documents in the dataset using either BM25 feature score or the relevance score of a regression model for the MSLR-WEB30K collection
Figure 5.13: Comparison of multiple data selection methods on the MSLR-WEB30K collection
Construction of Training Collection for Learning to Rank

Figure 5.14: Comparison of multiple data selection methods on the Yahoo! LTR collection
6 Conclusion

In my dissertation, I presented detailed and systematic studies of the properties of training collections for learning to rank. I also evaluated a number of training collection strategies and compared them extensively to existing baselines. The problems under examination are motivated by the need to construct effective and efficient training collections for (1) use in deployed search engine algorithms and (2) to advance the development of new algorithms.

One of the main conclusions of my dissertation is that a mixture of relevant and diverse documents leads to most effective training collections. This finding is supported by a number of experiments from Chapter 4 and Chapter 5 with actually constructed training collections.

The problem of how to define diversity between documents does not have a unique solution. Taking as diverse both a set of random documents and a set of documents that are maximally spread out may work well. My findings show that the set of random documents works well only in combination with relevant documents, while the set of maximally spread out documents works well independent of the presence of relevant documents. The definition of diversity as maximally spread out documents is motivated by the theory of Optimal Design of Experiments and depends on the learning algorithm.

I applied the theoretical framework of ODE to the problem of effectively constructing efficient training collections because it had a number of appealing properties such as sound theoretical foundation and the possibility to adapt the theory to the learning to rank problem. One conclusion from ODE is that the right definition of diversity between documents is influenced by two properties of the learning algorithm: the loss function of the learning algorithm and the hypothesis space. My experiments have produced results that are consistent with this theoretically derived conclusion. For example, Ranking SVM and linear regression
both express document scores as a linear combination of features, but differ on
the loss function. However, RankingSVM was very effective on training datasets
that emphasize relevant documents while linear regression was less effective on
this type of dataset. On the contrary, LambdaMart was more effective on the
datasets created by Optimal Design of Experiments than RankingSVM with a
linear kernel, even though both methods use similar loss functions. LambdaMart
is a combination of decision trees while RankingSVM, as used in my experiments,
employs a linear combination of features. The diversity of features turned out to
be more important for non-linear models as they have higher learning capacity.

After the extensive study of training collections, the original research questions
posed in the Introduction can be revisited.

6.1 Training Datasets Independent of the
Learning to Rank Algorithm

Can a dataset for learning to rank be created that is superior to any other
dataset independent of the learning method used? According to the theory of
ODE, better datasets can be constructed when specific properties of the learning
to rank algorithm are exploited. The theory shows that the best dataset is al-
gorithm dependent. As predicted by the theory, my experiments did not show a
data selection method that was superior across all learning to rank algorithms.
However, my experiments showed that there are training collections that perform
reasonably well for a number of learning to rank algorithms. For example, the
dataset I constructed with Optimal Design of Experiments with the constraint
of using top-ranked documents performed well for a number of learning to rank
algorithms. The datasets that combined random and top-ranked document also
performed very well.
6.2 Difference between Training Collections and Test Collections

My dissertation confirms that training collections are quite different from test collections. One difference is that a number of ranking algorithms are required for creating test collections. In contrast, effective training collection can be created without the use of many ranking algorithms or even without the use of any ranking algorithm. Traditionally test collections were created with multiple documents per query and only a few queries. Training collections work best with a large number of queries and a few documents per query. The properties of features are not used directly in the construction of test collections, while feature variability is an important dimension in training collections.
Bibliography


