Final Report

Research Project

Learning to rank for document retrieval in the context of web-search

May 1, 2019

University Of Exeter
College of Engineering, Mathematics and Physical Sciences

I certify that all material in this dissertation which is not my own work has been identified.

Abstract

"Learning to rank refers to using machine learning techniques in a ranking task." [1] Learning to rank is useful for applications in information retrieval, natural language processing, data mining, machine translation, computational biology and recommender systems [2]. In this project, we look at document retrieval, a branch of information retrieval, in the context of web-search. However, the techniques used in this report could easily be applied to other applications. Because of the importance of document retrieval, especially in the context of web-search, a large amount of research and development has gone into developing ranking algorithms which rank documents based on document features to indicate the importance and relevance of a document with respect to a query. This report summarises the research we have undertaken in this area, where we focus on comparing the performance of different ranking algorithms proposed in the literature and we introduce a new evolutionary algorithm approach to ranking, which we have named *RankEvolved*.

Contents

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 The Research Problem . . . 2
- 1.3 Objectives . . . 2

2 Summary Of Literature Review And Project Specification **3**
- 2.1 Learning To Rank Approaches . . . 3
- 2.2 Relevance Judgements . . . 3
- 2.3 Evaluation . . . 3
- 2.4 Evaluation Procedure . . . 4
- 2.5 Research Hypothesis . . . 4
- 2.6 Requirements . . . 4

3 Specification and General Design **4**
- 3.1 Project Design . . . 5
- 3.2 General Learning To Rank Design . . . 5

4 Investigating the different approaches **6**
- 4.1 Design And Development . . . 7
  - 4.1.1 Pointwise Approach . . . 7
  - 4.1.2 Pairwise Approach . . . 7
  - 4.1.3 Listwise Approach . . . 7
  - 4.1.4 Heuristic Ranking Model . . . 8
- 4.2 Results . . . 8
- 4.3 Conclusion . . . 9

5 Designing the evolutionary approach **9**
- 5.1 Design Overview . . . 9
- 5.2 Learning System Design . . . 10

6 Process, Development and High Level Design of the Evolutionary Approach **11**
- 6.1 Feature Extraction System Development . . . 11
- 6.2 Development of the Learning System . . . 12
  - 6.2.1 Initialisation . . . 12
  - 6.2.2 Fitness Function . . . 13
  - 6.2.3 Selection Method . . . 14
  - 6.2.4 Crossover Method . . . 15
  - 6.2.5 Mutation Method . . . 16
- 6.3 Development of the Evaluation System . . . 16
  - 6.3.1 Numerical Measures . . . 17
- 6.4 Testing . . . 18

7 Experimental Design **19**
- 7.1 LETOR . . . 19
- 7.2 Experimental Plan . . . 21
  - 7.2.1 Parameter Tuning . . . 21

7.2.2 Plan Of Experiments 22
7.3 Comparisons 22

8 Results And Evaluation **23**
8.1 Parameter Tuning Results 23
8.2 Official Results 23

9 Product And Fitness For Purpose **27**

10 Critical Analysis **28**
10.1 Limitations 29
10.2 Project Process And Approach 29

11 Future Work **29**

A Document Features **34**

B MQ2008Figures **35**

1 Introduction

With the development of the World Wide Web, we have created a vast amount of data. It is currently estimated that Google alone contains at least 46 billion web pages [3]. To navigate data of this magnitude it is necessary to have a well designed sorting system in place so that relevant web-pages or documents can be found. The process of finding these documents is called *document retrieval* (a branch of *information retrieval*). The document retrieval systems in place for finding the relevant documents are search engines, such as Google, Ask and Bing. The importance and use of search engines is immense, with there being over 3.5 billion searches per day and 1.2 trillion searches per year worldwide [4].

A simplified search engine is composed of a web crawler, a database and a search interface [5]. The database is created by WebCrawler's which use links to "travel" from document to document across the internet. For every document the WebCrawler's visit, they collect information about the features of the document and store these document features in a database. The search interface can then be used to input queries, where he query's keywords are looked up in the database to find documents related to the query. The search engines *ranker* sorts these documents using some algorithm which takes the document features that the WebCrawler obtained as inputs. The ranker aims to put the most popular and relevant documents with respect to the query at the beginning of the list and the least popular and relevant documents at the end of the list. Because of the importance of the ranker in search engines, much research has gone into the development of these ranking algorithms and it is these ranking algorithms that will be the focus of this research project.

The ranker or ranking model is an algorithm that uses document features to rank the relevance and importance of documents with respect to a query. Typically there are considered to be two categories of ranking models; relevance ranking models and importance ranking models.

Relevance ranking models aim to rank a list of documents based on how relevant the document is to the query. The relevance ranking models take an individual document as input, calculate the relevance score of the document to the query and then these scores are sorted in descending order. A commonly used example of a ranking model is the BM25 model[6]. Importance ranking models rank documents based on their own importance, which is closely linked to the document's popularity. The most popular importance ranking model is PageRank[7].

Given these ranking models and the many others that exist, it was hypothesised that there should be a way to combine these models to create an even more effective model. This can be done using machine learning, a subfield of computer science which can be defined as the study of the processes used to solve a problem by making use of some dataset to produce algorithms and statistical models [8]. By using machine learning we can not only combine these models but we can also combine other features collected by search engines to create a more effective ranking model automatically. When machine learning techniques are used to rank documents, it is known as **learning to rank**.

Learning to rank is a supervised learning task, which means we require training data and test data. The training data consists of queries and their corresponding documents, as well as the relevance judgments which is the relevance score of the document with respect to the query and is often assigned by human rankers. The documents are represented by feature vectors, $\mathbf{x}$. Where each feature represents some useful relevance or importance criteria such as the score given by a relevance ranking model.

1.1 Motivation

Having a good ranking algorithm is very important for search engines, as it effects the lives of millions, or even billions of people. If a ranking algorithm doesn't perform well, it would take its users longer to find useful documents, resulting in billions of hours effectively being wasted. Additionally learning to rank is not just limited to document retrieval and are useful for any application in which ranking is required such as online advertisement, personalized search, machine translation, document summarisation [2]. Therefore improving the ranking algorithms

in document retrieval will also improve many other areas.

In addition currently search engines ranking functions could be significantly improved upon, although search engines such as Google are very secretive about there exact ranking factors and algorithms, Bing has acknowledged that they use a learning to rank algorithm based on *RankNet*[9]. Which could well be improved upon.

1.2 The Research Problem

This project takes the form of research, where we attempt to answer a simple but important question in learning to rank: **"How do we improve on the ranking accuracy of current learning to rank algorithms?"** In other words, how can we use machine learning techniques to improve the ordering that documents are given in when we make a web search.

1.3 Objectives

As discussed in the literature review and project specification released before this report, we adopt an experimental approach in the undertaking of this project. In this project the aim is to research, implement and create ranking algorithms, which use document features in order to rank documents in response to queries.

Our first objective is to compare the different learning to rank techniques proposed in the literature so that we can assess which approach work best, as well as increasing our understanding of how they are implemented. This is necessary as there is little comparison done in the literature and their is no clear best approach. Therefore we will implement some of these different techniques and compare the results. Once we have made our comparisons, based on our findings we can carefully consider which approach is worth investigating further. Additionally this comparison will serve as a useful, and well needed resource for other researchers investigating learning to rank techniques.

Our second objective is to develop a new learning to rank algorithm as a means of demonstrating a solution to the research problem. At the time of writing we have completed the first objective and we have decided to implement a listwise approach which uses an evolutionary algorithm as the learning system. This is particularly interesting as its a relatively unexplored research area in learning to rank, with as far as we are aware, only 2 algorithms being proposed, despite both of these papers showing a lot of potential. In this report we describe the design and development of this approach, as well as the results achieved in doing so.

2 Summary Of Literature Review And Project Specification

Before the writing of this report, a literature review and project specification document was written, we summarise its contents in this section.

2.1 Learning To Rank Approaches

The three types of learning to rank approaches proposed in the literature are **pointwise**, **pairwise** and **listwise**. There full descriptions are given in the literature review and project specification and also in section 4 of this document.

The pointwise approach tends to be the simplest to implement as it transforms the problem into a regression, classification or ordinary regression problem, where we simply try to match feature vectors to targets, which we can use existing techniques to solve. The main disadvantage to this approach is that it only considers mapping a document to a relevance degree, so its learning system never considers the relative order between the documents, and so is likely to perform slightly worse for the purpose of ranking.

The pairwise approach tends to be the most popular approach it; its main advantage being that it considers the relative order between documents, by considering the ranking of a pair of documents, unlike in the point-wise approach. However a disadvantage to this approach is that it is more affected by poorly labelled documents (documents given the wrong ground truth relevance) as if just 1 document is mislabeled it will result in lots of mis-labelled document pairs.

The advantages to the listwise approach are that it represents the ranking process in a more natural way, whereby the input space is the set of documents associated with a query and the output space is the full ranked list. This approach allows the relative order between documents to be considered even more than in the pairwise approach. The main disadvantage of this approach is that it has very high complexity as the loss function must be capable of evaluating the full ranked list. Additionally this approach tends to be the most difficult to implement as we can't make direct use of existing machine learning techniques.

2.2 Relevance Judgements

Ground truth relevance judgements are an essential part of our training data; There are multiple ways of representing and generating relevance judgements. We can represent them by their relevance degree, their pariwse preference and as the total order of the correctly ranked documents. In the literature review and project specification we decided to use the relevance degree, which is where each document is given a label, which can be from the set {perfect, excellent, good, fair, bad}. Binary relevance degrees such as ones from the set {relevant, irrelevant} are also commonly used. Alternatively, any number of different grades would be acceptable for use.

Currently, there are two ways of generating the relevance degrees; the first is using humans to label how relevant a document is and the second is using clickthrough data. Clickthrough data is collected by search engines and records the documents the user clicks on after entering a query as well as other information such as the time spent on that site. We will be using relevance degrees generated by human rankers in this project.

2.3 Evaluation

Evaluating exactly how well a ranking algorithm performs is of optimal importance for this project, as we aim to achieve the highest possible ranking accuracy. Several statistical methods exist to compare how close the ordered list generated by the ranking algorithm is to the true ranking. For example MAP[10], or mean average precision which is used when using binary relevance degrees (relevant or irrelevant) and NDCG[11] or normalised discounted cumulative gain which is an evaluation measure that is capable of using non-binary relevance degrees such as the 5

graded relevance degrees, which works based on the principle that highly relevant documents should be penalised more heavily the further down they are on the ranked list.

2.4 Evaluation Procedure

In the project specification part of the literature review and project specification we outlined how we will approach the problem.

This project first aims to implement existing learning to rank algorithms. Once we have achieved this, we will implement evaluation measures to assess how well each of the implemented learning to rank algorithms perform. This will allow us to compare learning to rank algorithms.

Once we have achieved the first part of the project can then attempt to change learning to rank algorithms and implement different ideas to assess whether it improves the ranking scores or not. After this experimentation, we will then be able to perfect our proposed learning to ranking algorithm and give our results. These results for our proposed algorithm can then be compared to several different algorithms to get an overall assessment of how well our algorithm performs.

2.5 Research Hypothesis

We initially hypothesise that the listwise approach performs the best, and the pointwise approach performs the worst, we will need to investigate this as there is little comparison done in the literature.

2.6 Requirements

A summary of the requirements given in the literature review include:

- Implement a pointwise approach
- Implement a pairwise approach
- Implement a listwise approach
- Implement a new approach based on what has been learned from the other approaches.

For each of these approaches we require:

- A feature extraction system, which converts the training data to a useable form.
- A learning system, which should create the ranking model.
- A ranking system, which orders the documents by the predicted relevance given by the ranking model.
- An evaluation system to measure of how well the learning to rank algorithm performs.

Once trained, they should take as input a query and a set of unordered documents related to the query. It should then output the ordered list of documents, where they are ordered based on relevance. The documents relevance is calculated by the ranking model, which is created by the learning system.

3 Specification and General Design

In this section of the report we begin by describing precisely the design of the project as a whole, and we will then list the requirements and design of the general learning-to-rank setup that we will use throughout this project. We find this is inline with the specification previously developed, but we have made a few minor modifications which we will discuss.

3.1 Project Design

This project first aims to implement existing learning to rank algorithms. We will implement an algorithm from the pointwise, pairwise and listwise approaches as well as a heuristic ranking model, BM25, to use as a baseline to compare its performance against learning to rank techniques. Once we have implemented these algorithms we must implement an evaluation measure to assess how well each of these algorithms perform.

When we have achieved the first part of this project we will hopefully be able to see which approach to learning to rank performs the best. We can then make modifications and implement different ideas to learning to rank algorithms of our chosen approach in an experimental setting to assess how these modifications and ideas impact performance. After this experimentation we will be able to propose a learning to ranking algorithm where we aim to achieve the best possible ranking accuracy. The scores can then be compared to several different algorithms proposed in the literature to asses how well our algorithm performs.

We have decided to modify the project design, in the project specification document previously produced we implied that we would make the focus of the report the comparison between the pointwise, pairwise and listwise techniques, with only a section that focuses on the implementation of our own algorithm. Instead this report mainly focuses on the implementation of our own algorithm, with only a preliminary section dedicated to the comparison of the different techniques. We do this because most of the specific algorithms that we have implemented in the comparison of approaches section of this report can be read and analysed in the literature. Furthermore the comparison of these techniques will not necessarily result in a best approach. However the algorithm which we propose is unique and challenging to implement and makes for a more interesting topic for this report. Additionally the results we have obtained show that this algorithm performs particularly well, and is more significant for the field of learning to rank.

We decide to use Python to implement these approaches we make this decision because of the large selection of libraries useful for machine learning, such as *sciKit-Learn*[12] , *SciPy*[13] and *numpy*[14] and the large number of frameworks such as *TensorFlow*[15] and *PyTorch*[16]. Additionally as Python is relatively easy to understand, it allows us to focus more on the design of algorithms rather than spending a lot of time dealing with syntax and the understanding of the code.

We write our code in a IPython/Jupyter notebook, this is particularly useful for research projects as the interactive environment allows us to combine different blocks of code execution depending on what exactly we wish to experiment with as well as allowing us to write detailed notes about what we are attempting, aiding the research development process. Additionally being able to get output easily allows us to test blocks of code, and test the performance of the algorithm all without having to leave the notebook.

3.2 General Learning To Rank Design

All the learning to rank systems that we will implement have a relatively similar design, as such we generalise this design in the section of the report, we will go into the specific high-level design for our implemented design later.

Learning to rank is normally a supervised learning task. Supervised learning is used when we have input variables x and target values y. We then attempt to learn a function to map input variables to targets. Once the mapping function has been learnt, it can be used on new data which has not used in the learning process, to predict the value of y.

In learning to rank our dataset consists of a list of labeled documents, $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i$ represents one of N feature vectors representing the document and y_i represents the label of this document, which in this case corresponds to the relevance judgement of the document. The documents feature vector $\mathbf{x}_i$ of dimension D, correspond to a query; where each feature $x_{ij}, j = \{1, ..., D\}$ represents some useful relevance or importance criteria with respect to the query such as the number of query keywords present in the document, see FIGURE.

2 example rows from the MSLR-WEB10K dataset[17] showing 136-dimensional feature vectors are given below:

0	*qid:1*	*1:3*	*2:0*	*3:2*	*4:2*	*...*	*135:0*	*136:0*
2	*qid:5*	*1:0*	*2:1*	*3:4*	*4:3*	*...*	*135:1*	*136:0*

Here the first number in the row corresponds to the relevance judgement or relevance degree of the document, this is the label of the feature vector that we are trying to predict and it represents the ground truth of how relevant a human ranker believes the documents is with respect to the query. The second number proceeded by 'qid' is the query ID of the document, which is the query that the document corresponds to. The other numbers represent the feature IDs and the feature scores *(featureID:FeatureScore)*. In this particular dataset we have 136 different document features, each feature represents some useful relevance or importance criteria such as the score given by a relevance ranking model.

The relevance judgement can be generated in several different ways and can be represented in several different forms as discussed in the literature review. The most popular form is where each document is given a label, commonly these labels are assigned from the set perfect, excellent, good, fair, bad; which correspond to the numbers $\{4,3,2,1,0\}$ respectively. For example if a document is perfectly relevant to the query it will have a relevance degree of 4.

In some approaches to learning to rank such as the pairwise and listwise techniques the training data is transformed, so that the relative order between document feature vectors, calculated using the relevance degrees are considered as the training data. We will describe these cases in the next section of this report.

We will require some way of extracting the features, relevance judgements and queries from the text based training data, as such we will need a *feature extraction system*, this should convert the features into a form that we can use more easily, such as an array or list. It is also be necessary for us to sort the documents so that they are grouped into their respective queries.

We then need to create a *learning system*. A learning system, of which there are many different types, uses the training data to learn the best way of combining all the document features in order to create a *ranking model* that can directly predict the relevance judgement (pointwise) or predict relative order between documents (pairwise and listwise), as accurately as possible. The exact design of the individual learning systems implemented in this project will be described in the relevant sections.

Once the training data, learning system and ranking model are in place, the test phase can begin. For each piece of test data consisting of the query and its corresponding documents represented by a feature vector, the ranking model gives each document a score and the *ranking system* then sorts the documents according to these scores to give the most relevant documents at the beginning of the list, and the least relevant documents at the end of the list. The exact ranking model depends on the learning to rank approach used.

Once we have the algorithm in place, we require an *evaluation system*. This system should take as input a test query with their corresponding documents and should output a numerical measure of how accurately the ranking system sorts the document based on the known order. We then average this numerical measure across all of these test queries, to get the final measure. There are a range of numerical measures that we can choose from such as *NDCG* (normalised discounted cumulative gain) and *MAP* (mean average precision). We will discuss these evaluation measures more in subsequent sections.

4 Investigating the different approaches

In this section we investigate the different learning to rank approaches to attempt to determine which approach is worth investigating further. This is part of the research based development process for our own learning to rank system that we have created, and is an essential part of the overall project as it is based of these results that we decide which learning to rank approach we wish to investigate further. Additionally the understanding gained by implementing these will help us when developing our own approach. We do not intend for this to be the main topic of this report as such we keep the design and development of these approaches brief.

4.1 Design And Development

In this section we briefly describe the exact learning system implemented for each of the pointwise, pairwise and listwise approaches.

4.1.1 Pointwise Approach

In the pointwise approach, the aim is to transform the ranking problem into a classification, regression or ordinary regression problem, for which we can use the techniques used to solve classification, regression and ordinary regression problems to act as our learning system. The training data is then typical supervised learning data, where we are trying to map document feature vectors $\boldsymbol{x}$ to its relevance degree y.

In this implementation we decide to transform the problem into a regression problem, and we decide to use linear regression to act as our learning system. Linear regression uses a linear model to attempt to predict the relevance degree y for each document, using the documents features. As we have multiple document features it is necessary for us to use multiple linear regression. We represent our model with:

$$y = w_0 + w_1 f_1 + w_2 f_2 + ... + w_n f_n,$$

where f are the feature values, and w are the weights or constants that we wish to determine in order to predict the relevance degree y as accurately as possible.

To determine how well a set of weights predicts the relevance degree we need a loss function, in this case we use the mean squared error (MSE) as the loss function. The mean squared error measures the squared difference between the actual relevance degree and the predicted relevance degree, we then average this loss all documents. We calculate it using:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 f_1 + ... + w_n f_n))^2,$$

where n is the number of documents, y_i is the actual relevance degree. We then use gradient descent SITE to minimise the mean squared error.

When implementing this we make use of Pythons sciKit-Learn package, to perform linear regression for us.

4.1.2 Pairwise Approach

The pairwise approach differs from the pointwise approach in that it does not try and predict the exact relevance judgement for a document; rather it attempts to find which document out of a pair, is more relevant to the query. The ranking problem for the pairwise approach is normally reduced to a classification problem where we must classify which document is preferred. The aim of the learning system is to minimise the number of inversions, that is to say to minimise the cases where we hypothesise that $\mathbf{x}_{i,j}$ (where i is the query and j is the document) is more relevant than document $\mathbf{x}_{i,k}$ when in fact the opposite is true according to the ground truth relevance judgments, $y_{i,k} > y_{i,j}$.

For the pairwise technique we choose to implement RankNet[9], we choose RankNet as it is a learning to rank algorithm that is currently being used by search engines [18]. In RankNet a neural network is used as the ranking model, then the loss function aims to minimise the number of inversions of a preference pair, this is done using gradient descent. Credit to RankLib library [19] for the implementation.

4.1.3 Listwise Approach

The listwise approach works differently to the other approaches in that it adopts an approach more natural to the concept of ranking. The training data consists of all documents associated with the query, and the relevance

judgments of documents. The relevance judgments are transformed into an ordered list such that for every document associated with query i, for all $y_{i,k}$ satisfying $y_{i,j} > y_{i,k}$ we have $\pi_y(j) > \pi_y(k)$ where $\pi_y(j)$ represents the position of document j in the ordered list y. The learning problem then becomes sorting the documents into an ordered list which is as close to the ground truth ordered list. In this way the exact values of the predicted relevance degrees do not matter, only the relative order of these documents do. There are no existing machine learning techniques to be used in this case; hence this approach can be the most challenging approach to implement.

We decide to implement AdaRank[20]as the learning algorithm. Adarank repeatedly constructs *'weak rankers'* on the basis of re-weighted training queries, it then combines these weak rankers linearly to make predictions on the data. Adarank makes use of the numerical measures all ready in place such as NDCG or MAP. As such depending on what numerical measure is used in the training process, the results we obtain tend to be slightly different. Credit to RankLib library [19] for the implementation.

4.1.4 Heuristic Ranking Model

An example of a heuristic ranking model is the BM25 model [21]. It is based on the probability retrieval framework developed in the 1970's and 1980's [22]. The BM25 model works by counting the number of appearances of the query's keywords in the document [23]. For example, if the search query is "Queen Elizabeth" the keywords would be "Queen" and "Elizabeth", and the number of times "Queen" and "Elizabeth" appear in a document is counted.

The BM25 scores can then be calculated when given a query Q, containing keywords $q_1, \ldots, q_n$ and a document D, using the following formula:

$$score(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})}$$

where $f(q_i, D)$ is q_i's term frequency in document D, $| D |$ is the documents length in words, avgdl is the average document length for all the documents that are in the document collection, k_1 and b are free parameters $k_1 > 0, 0 < b < 1$ and IDF(q_i) is the inverse document frequency weight of q_i, it's calculated by:

$$IDF(q_i) = \log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

where N in the total number of documents in the collection, and $n(q_i)$ are the number of documents containing q_i.

Implementing this model is not difficult as its one of the features given in the data, as such to achieve results for this model all we must do is sort documents by the already given BM25 scores.

4.2 Results

Here we compare the results of the pointwise, pairwise and listwise algorithms using the MQ2007 dataset [24]. For the full design of these experiments and the numerical measures see Section 7 and 6 respectively. However for now it is enough to know that the higher the NDCG, MAP and Precision scores the better the algorithm performance.

Algorithm	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	MAP
AdaRankMAP	0.4392	0.4301	**0.4230**	**0.4126**	**0.4054**	0.3953	0.3873	0.3823	**0.3782**	**0.3738**	**0.4577**
RankNet	**0.4515**	0.4303	0.4129	0.4004	0.3915	0.3863	0.3791	0.3725	0.3665	0.3604	0.4500
LinearRegression	0.4356	**0.4350**	0.4204	0.4114	0.4042	**0.3960**	**0.3890**	**0.3832**	0.3770	0.3723	0.4497
BM25	0.3741	0.3659	0.3596	0.3521	0.3477	0.3429	0.3410	0.3370	0.3348	0.3312	0.3941

$NDCG_1$	$NDCG_2$	$NDCG_3$	$NDCG_4$	$NDCG_5$	$NDCG_6$	$NDCG_7$	$NDCG_8$	$NDCG_9$	$NDCG_{10}$	$NDCG_\infty$
0.3821	0.3900	**0.3984**	**0.4024**	**0.4070**	**0.4113**	**0.4154**	**0.4217**	**0.4278**	**0.4335**	**0.4891**
0.3893	0.3895	0.3907	0.3924	0.3966	0.4038	0.4087	0.4136	0.4192	0.4235	0.4821
0.3752	**0.3904**	0.3935	0.3979	0.4031	0.4079	0.4121	0.4180	0.4227	0.4288	0.4845
03174.	0.3234	0.3312	0.3358	0.3397	0.3451	0.3523	0.3582	0.3648	0.3699	0.4329

4.3 Conclusion

We can see that for the limited algorithms considered here, the listwise approach appears to perform the best. This seems to support our initial hypothesis and is also supported in literature. Therefore we decide that we wish to try and further improve upon different listwise techniques, in hopes of obtaining results that can compete with the current top algorithms in order to demonstrate that using listwise techniques is the best way of improving ranking accuracy. We note that we have only considered a limited set of algorithms here, therefore we can not conclude based on this sample of algorithms that listwise is the best approach, however we can conclude that algorithms of the listwise approach has the capability to perform very well, which is adequate for our purpose.

5 Designing the evolutionary approach

In this section we give the design of the learning to rank system that we are implementing and give the detailed design of the learning system, as it is the learning system in particular that makes our learning to rank system unique. As part of our research methodology we have attempted several different designs in this project however we focus here on the final chosen design, an evolutionary algorithm approach. Here we specify exactly what is required from our system, further higher level design is given in the next section.

5.1 Design Overview

We have already given the general design of learning to rank methods in section 3, we briefly summarise these and add some more details specific to this approach.

For the feature extraction system we need to have a system that will take as input the text based training data, which then converts this training data into a form more accessible for our system. As part of this we will put all our documents into a list, where each document is a the list of features, without the feature ID being included. Additionally we will sort all the different documents into their respective queries. Once the feature extraction system is implemented, the learning system will be able to access easily our features, relevance judgments and query ID's .

For the learning system we decide to use an evolutionary algorithm. The learning system uses the training data to learn the best way of combining all the document features in order to create the ranking model. We decide to use an evolutionary algorithm as our learning method as its a relatively unexplored area in learning to rank despite it having the potential to perform very well.

The ranking model takes the values of a documents features, combines them in some way, and outputs the predicted relevance judgement. We propose a simple linear function as the ranking model which will take the form of

$$y = \alpha_1 f_1 + \alpha_2 f_2 + ... + \alpha_n f_n$$

where f are the feature values of the document, n is the total number of features and α are the constants or weights that we wish to determine in order to predict a relevance degree y. It is important to note that because this is a listwise approach the exact value of y does not have to match the ground truth relevance degree, only the relative order between the predicted relevance degree of different documents has an effect. We decide to use a linear ranking

model as it is the most common technique used in learning to rank and has been shown to produce good results. Additionally for the purpose of an evolutionary algorithm it is a lot easier to implement.

The ranking system sorts the documents according to their relevance scores so as to give the most relevant documents at the beginning of the list and the least relevant documents at the end of the list. We will need the ranking system a lot in our design as we will not only need to use it for the test data, but we will also need to use it when training the learning system as a way to asses how well the proposed linear ranking model is working. For this reason we will combine the ranking system with the evaluation system.

The evaluation system should take the full set of test query's with their corresponding documents as the input. We then design the evaluation system so that it makes use of the ranking model to score the documents; the documents are then sorted by these scores. The ground truth relevance degrees are also given and the evaluation system should output a numerical measure, which assesses how accurately the ranking system sorts the document based on the ground truth list. We will then average this numerical measure across all of these test queries, to get a final assessment of how well the algorithm performs.

To summarise we first design a way to extract the features from the textual input and sort them by there queries, we then use the evolutionary algorithm to create a ranking model that will be used to rank the documents in order from most relevant to least relevant with respect to each query. Finally the overall performance of the algorithm is evaluated by the evaluation system.

5.2 Learning System Design

In this section we give the design of the learning system, for which we have decided to use an evolutionary algorithm. We hypothesize that evolutionary algorithms perform well because they are a listwise approach, which we have shown have the capability of performing well. In addition, with an evolutionary algorithm using the listwise approach, we can make direct use of the numerical measures that are used to test the performance of the overall system, as a training tool for the learning system.

We decide to use an evolutionary algorithm as the learning system. Evolutionary algorithms are modelled on biological evolution (natural selection), where the mechanisms in place in biology, such as reproduction, selection and mutation can be modelled and implemented in code. This works by initialising many candidate solution to a problem, then these candidate solutions are assessed in some way; the solutions that do badly are removed, whilst the solutions that do well are allowed to"breed" and create new individuals to replace those removed, thus improving the 'gene pool'. These new individuals may also be mutated on, to ensure diversity in the population.

The basic framework for an evolutionary algorithm is given in Algorithm 1.

Algorithm 1 Basic Evolutionary Algorithm Framework

```
Initialise Population
while Iteration <= Max Iteration do
    Iteration = Iteration + 1
    Evaluate Population
    Select Best Performing Individuals
    for Best Performing Individuals do
        Perform Crossover
        Perform Mutation
    end
    Select New Population
end
```

This means the learning system will require:

- An initialisation method, which should randomly create individuals to make up the population. In this project we plan to use the ranking model as the individual, where we can change feature constants in order to change how well the ranking model ranks all the individuals.
- An evaluation method which should be capable of assessing how well an individual performs, this is known in the literature as the fitness function. The fitness function takes an individual as input and outputs a numerical score based on the individuals performance, a individual with a high fitness value means the individual performed better than an individual with a low fitness value. We can conveniently make use of existing statistical evaluation measures, that have been developed to assess how well a list has been ranked as our fitness function.
- A selection method, this is responsible for deciding which best performing members of the population are allowed to reproduce by modelling genetic techniques, this has the effect of removing the individuals from the population that perform badly and only keep those that do well.
- A crossover method, this should be responsible for combining together 2 (or possibly more) individuals, to produce a new individual, modelling genetic reproduction.
- A mutation method, this should take an individual in the population as its input, modify the individual slightly, and output the new mutated individual.
- Finally we will need to decide which individuals from the existing population get replaced, and which if any are allowed to remain.

We will provide the high level design of these methods in the following section.

When deciding the exact techniques to consider an important concept to be aware of is genetic diversity. A population that is more diverse is more likely to provide a solution that the finds the global optimum (the best possible ranking). If the population is not diverse, especially for the first few iterations of the algorithm, all the individual ranking functions will be very similar to each other, which means a possible best solution is likely to be missed and continued iterations will only result in finding a local optimum. Therefore we attempt to design our algorithm to converge on its solution as slowly as possible, maintaining the diversity of individuals.

6 Process, Development and High Level Design of the Evolutionary Approach

In this section we outline the process of development of this report, as well as some higher level design. As this is a research project we don't focus on a development methodology, instead we discuss the choice of our techniques that we use, and the details of how we implemented them. We note that the development process did not just consist of developing the evolutionary algorithms, several different ideas where attempted including implementing several different existing algorithms and experimenting with several different loss functions, however we decide to focus on the development of the evolutionary approach, as its findings are more significant.

6.1 Feature Extraction System Development

The development of the feature extraction system can be split into 2 parts. Firstly we read the file extracting the relevant information and putting this information into numpy arrays when possible, else we put them into python lists. We originally decided to use numpy arrays for all data storage due to the increase in performance of numpy arrays compared to lists, however this became problematic because of the high degree of data-structure required for this project and we occasionally have lists containing different data types and more complicated data structures such as tuples, which numpy arrays do not allow; hence we use some Python lists.

When we first read a line from the text file we have a string of the form:

'0 qid:134 1:5.4321 2:0.5432 3:2 4:2 ... 135:0 136:0'

This is not ideal for several reasons, firstly we would prefer to have a list of features rather than a full string and secondly a large amount of this data, such as the *qid* and the feature identification numbers are redundant as we only require the qid number and feature values. Therefore we transform the above document into a more useful form, where all items in the list are floats or integers:

[0, 134, 5.4321, 0.5432, 2, 2, ..., 0, 0]

To do this we make use of pythons *string.split(separator)* function, where we can split a string into a list. We also decide to swap the order of the relevance degree and the query ID to make it easier to sort by query ID in the next part of the feature extraction system.

[134, 0, 5.4321, 0.5432, 2, 2, ..., 0, 0]

We perform this step for each document in the file, so we end up with a list containing all the documents, represented by a list of features (with the queryID and relevance degree).

When testing this system for different data sets, we had a problem as some data sets represent documents of a different form, for example, in the MQ2007 dataset an example document takes the form:

'0 qid:134 1:5.4321 2:0.5432 3:2 4:2 ... 45:0 46:0 #docid = GX000-00-00 inc = 1 prob = 0.0246'

The different number of features is fine, as our system accounts for that, however the extra text on the end such as the #docid is problematic as they get classified as features. As such we must remove these from the string, we do this simply by checking if *#docid* is present in the document string, if present we stop adding features to the list at that point. This is an adequate solution as there are only a few different data sets that we use in this project, however if we wish to add a new learning to rank data set we may have to adapt the feature extraction system.

For the second part of the system we sort the list of all documents (which themselves are lists) into query specific lists, where documents which correspond to the same query are grouped together in a list, as such we are left with a list of lists of list of lists:

ListOfQueries = [[[document],[document],...],[[document],[document],...],...]

Where we have a list containing all queries, each query then contains a list of all the documents and the documents are the list of all the features, (along with the relevance judgement). We note that we have removed the queryID from the document list as it is no longer required.

When implementing this we encountered several problems when reading the file. Firstly because of the size of the training data we found that our computer didn't have enough RAM to read all the data in one go, as such we had to split the training data into sections and read them individually. Secondly we originally used numpy to read from the file however this ran into problem in some version of Python as the strings were often modified by numpy to include the byte representation of the data at the start of each string, as such we abandoned using numpy as the method of reading from the file.

6.2 Development of the Learning System

6.2.1 Initialisation

For the initialisation part of the learning system we are tasked with implementing several different linear ranking models (ranking functions). We choose to represent these ranking functions with the list of feature constants $[\alpha_1, \alpha_2, \alpha_3, ..., \alpha_{n-1}, \alpha_n]$ where n is the number of features. Then for each document we can calculate the relevance

given by the ranking model by calculating $y = \alpha_1 f_1 + \alpha_2 f_2 + ... + \alpha_{n-1} f_{n-1} + \alpha_n f_n$, where f_n represents the nth document features value. For each of the feature constants we initialise their value by randomly generating a float uniformly in the semi-open range $[-1.0, 1.0)$. Initially we decided to generate feature constants in the range $[0, 1.0)$ (and not allow for negative values to ever be a feature constant) however this was a problem as the results obtained were not as good as we would have liked, to resolve this problem we believe it better to include negative values as it is possible that for some features having a high score, actually effects the relevance degree negatively (such as the number of slashes in the URL), as opposed to not at all as previous assumed.

6.2.2 Fitness Function

The fitness function gives a score for each ranking function based on how well it ranks the documents compared to the ground truth order of the relevance judgements. We do this for the documents in every query, and it is the average $NDCG_{10}$ across all the queries that we use for the fitness. We will discuss more about numerical measures and $NDCG_{10}$ in *section 6.3.1*.

For the fitness function the evaluation system is set the task of assessing how well a ranking function performs. To do this it takes the full set of query's and their corresponding documents and the ranking functions as input. Then for each ranking function, for each query, for each document it uses the ranking function to calculate the assigned relevance score. Each query then sorts all its documents by these scores. Once this has been achieved the ground truth relevance degree is extracted from each document, whilst maintaining the order of the sorted documents, this results in a list of ground truth relevance degrees where the ordering is decided by the predicted relevance degree calculated by the ranking function. Once we have this list for a query we use a numerical measure to calculate the accuracy of the list. For example if the list begins with $[4,4,4,4,3,3,3,3,3,3,...]$ our ranking function has performed perfectly and has managed to predict correctly the order of relevance of the documents (for the first 10 terms). There are several numerical measures in place that can give a numerical score to the accuracy of the list, these include *DCG*, *NDCG*, Precision and Mean Average Precision. The numerical score for each of these measured in then averaged across all the queries to give the final assessment of how well the ranking function performs. The algorithm for the fitness function is shown in Algorithm 2.

Algorithm 2 Fitness Function

```
Input 1: List of ranking functions
Input 2: List of queries containing their corresponding documents
for Ranking Function do
    for query do
        for document do
            Use ranking function and document features to calculate predicted relevance degree
            Get ground truth relevance degree from the document
        end
        Sort all documents by predicted relevance degree in descending order
        Create list of ground truth relevance degree based on the order of predicted relevance degrees
        Calculate the NDCG10 score of this list
    end
    Average NDCG score across all queries use this as the fitness value
end
Sort the ranking functions in descending order of the average NDCG scores
```

The algorithm for the fitness function is quite challenging to implement as such we decide to split it up into 3 separate functions in the code.

6.2.3 Selection Method

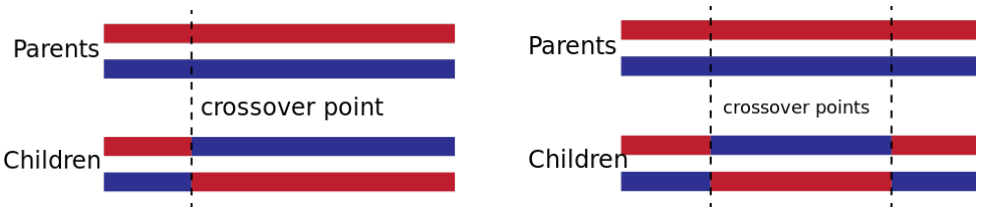
When selecting the next generation we decide to keep the top individuals of the previous generation unmodified in the next generation to ensure we never loose the best solution and to allow our algorithm to be stopped at any time and to guarantee good solutions, additionally some new individuals are created by only mutating a single parent. The rest of the individuals in the next population are generated by crossover followed by mutation. We have decided to select which individuals in the population will become parents for the population based on tournament selection[25].

Tournament selection decides which individuals in the population will become a parent for the next generation, it works by randomly choosing k individuals from the population, it then chooses the individual with the best fitness out of these k chosen individuals to become a parent; each required parent is decided in this way. We have decided to use tournament selection because many of our fitness values generated by *NDCG* are very similar in score, particularly in later generations. This means that other selection methods such as roulette wheel selection[26], where the fitness value is proportional to the chance of being chosen does not work well, as every individual ends up having almost the exact same chances of being chosen. As such selection methods based on the fitness ranking rather than the fitness value should be used, therefore tournament selection makes a good choice. We summarise which new individuals we use in algorithm 3.

Algorithm 3 Selecting New Individuals

Input 1: Sorted Ranking Functions
Input 2: Mutation Rate
Input 3: Percentage Of Population To Go In Next Generation Unmodified (UNMODIFIED PERCENTAGE)
Input 4: Percentage Of Population That Is Used For Only Mutation (MUTATION PERCENTAGE)
Create new empty population
for *Top UNMODIFIED PERCENTAGE of individuals* **do**
    Add to the new population
end
for *Top MUTATION PERCENTAGE of individuals* **do**
    Mutate individuals
    Add to the new population.
end
for *Number of individuals needed to fill population* **do**
    Decide parent 1 by performing tournament selection
    Decide parent 2 by performing tournament selection
    Crossover parent1 and parent 2, producing 2 individuals
    for *Both individuals produced by crossover* **do**
        if *Randomly chosen number between 0 and 1 is less than 0.9* **then**
            Perform mutation
            Add new individual to the new Population
        end
        else
            Add new non-mutated individual to the new population
        end
    end
end

After experimenting with this system we found that our results were often very unpredictable sometimes the algorithm performed very well and sometimes not. We concluded that the reason for this is that often our system



(a) Single-point crossover (b) Two-point crossover

Figure Set 1: images produced by [27].

finds a local optimum as the solution rather than a global optimum. To attempt to resolve this problem we decide to modify our original selection algorithm to attempt to increase the diversity of individuals, to do this we decide to only keep the single best individual in the next generation from the previous, rather than several. For the same reason we decide not to mutate individuals from a single parent, instead all new individuals should be generated by crossover always followed by mutation.

6.2.4 Crossover Method

As far as we are aware genetic programming has not been attempted before for the learning to rank with the representation of individuals that we have decided upon, therefore we are not sure which crossover method will perform the best, in fact as far as we are aware there is no research done in the literature on the crossover functions in genetic programming for the purpose of learning to rank at all. Therefore we have decided to implement several different crossover functions and implement them all to investigate in an experimental setting which works better. This will not only be beneficial for our results, but will also help future work done in this area.

We have decided to implement:

- Uniform crossover
- Single-Point Crossover
- 2-Point Crossover

For our uniform crossover function, new individuals feature constants are independently chosen from the two parents. For example if one parent is represented by $y_1 = [\alpha_1, \alpha_2, \alpha_3, ..., \alpha_{n-1}, \alpha_n]$ and the other parent is represented by $y_2 = [\beta_1, \beta_2, \beta_3, ..., \beta_{n-1}, \beta_n]$ then a single feature constant such as c_1 in the new individual, will be randomly decided with equal probability between α_1 and β_1, an example child ranking function could be $y_3 = [\beta_1, \alpha_2, \alpha_3, ..., \beta_{n-1}, \alpha_n]$.

For our single-point crossover function we have randomly chosen a point on both parents ranking functions, known as the crossover point. We then combine the ranking functions of the two parents. The new individual takes the part of the ranking function to the left of the crossover point from one the parents, and takes the part of the ranking function to the right of the crossover point from the other parent, creating a new individual. Additionally we then create another new individual from the two parents, only for this individual we swap which parent is used for the ranking function to the left and right of the crossover point. We have summarised this in Figure Set 1a.

Our 2-point crossover function is very similar to the single-point crossover function, the only different being that we choose 2 points on both the parents ranking function to be the crossover points, where we ensure that the first crossover point is defined before the second crossover point. The new individual will then take the the middle section between both crossover points from one parent, and the end sections (from the beginning of the ranking function to the first crossover point, and from the second crossover point to the end of the ranking function) from the other parent. The second individual created from the two parents is the same only we have swapped which

parent is used for the middle section and which parent is used for the end sections. We have summarised this in Figure Set 1b.

6.2.5 Mutation Method

The mutation function takes a single ranking function as its input, and outputs a similar but different ranking function resulting in a mutation. We decide to mutate only some of the individual ranking function constants. We implement this by for each ranking function being generated, we generate a random number uniformly from the range [0,1.0) if this number is less than the defined *mutation rate* we decide to mutate it. This value will be chosen based on preliminary experiments. For the exact mutation method we have taken inspiration from another paper, ES-Rank[28] which uses a (1+1) evolutionary strategy, which means they use 1 parent that produces 1 offspring; due to the importance of the mutation method in such a strategy we decide to use the mutation method developed by them:

$$MutatedGene_n = Gene_n + Gaussian(0,1) * exp(Cauchy(0,1))$$

Here $Gaussian(0,1)$ corresponds to a Gaussian (normal) distributed number with a mean of 0 and a standard deviation of 1. Cauchy(0,1) corresponds to a cumulative distributed random number, this ensures that it takes a value between 0 and 1. We note that ES-Ranks mutation method consists of more than just the equation given above.

We have modified our design slightly in that we now implement an adaptive mutation rate, which if the algorithm has not improved in a number of iterations, the mutation rate is increased. We do this in an attempt to increase more diversity into the individuals allowing them to escape from any local optimums in the hopes of finding a global optimum.

6.3 Development of the Evaluation System

In this approach the evaluation system has 2 purposes.

- Firstly, it is used as the fitness function for the evolutionary algorithm, where the performance of each ranking function must be evaluated each iteration. In this case the full list of ranking functions must be used as the input of the evaluation system.
- Secondly, once the algorithm has been trained it must be used to evaluate how well the chosen ranking function performs on the test data.

We have described how it is used for the fitness function in the previous section, for the second of these purposes the evaluation system is used once the learning system is finished training to assess how well it preforms on the test data. This is a very similar process to calculating the fitness function, the main differences being:

- We are only allowed to use 1 ranking function to evaluate the data to ensure validity of our results. Depending on which dataset we are using we either choose the the ranking function that has the best fitness after the last iteration, to evaluate or if the dataset comes with validation set, we may perform model selection on the validation data to select the best ranking function out of the whole final population to evaluate. Often there are rules that we must follow in this last case, we will discuss this in more detail in the experimental design section.
- We calculate several evaluation measures as opposed to just *NDCG*. Calculating several measures is the standard in learning to rank, as it allows other researchers to compare your work to others in full. We discuss the different evaluation measures in the following section.

.

The algorithm for the evaluation system is shown in algorithm 4.

Algorithm 4 Evaluation System

Input 1: Ranking function
Input 2: List of test queries containing their corresponding documents
for *query* **do**
    for *document* **do**
        Use ranking function and document features to calculate predicted relevance degree y
        Get ground truth relevance degree from the document
    end
    Sort all documents by predicted relevance degree in descending order
    Create list of ground truth relevance degree based on the order of predicted relevance degrees
    for $n \in \{1, 2, ..., 10\}$ **do**
        Calculate the $NDCG_n$ score of the ground truth relevance degree list
        Calculate the $Precision_n$ score for the ground truth relevance degree list
    end
end
for $n \in \{1, 2, ..., 10\}$ **do**
    Calculate the mean $NDCG_n$ score across all queries
    Calculate the mean $Precision_n$ score across all queries
end
Calculate *MAP* which is across all queries

When we implement the evaluation system, we must ensure that we use the feature extraction system on the test data to ensure the evaluation system works correctly.

6.3.1 Numerical Measures

Precision and MAP(mean average precision) is used when using binary relevance degrees (where each document is either relevant or irrelevant). Although it is also possible to transform relevance degrees with more than 2 grades into binary relevance degrees, for example by letting a relevance degree of 0 mean irrelevant, and any relevance degree higher than that mean relevant. Precision, P is the proportion of total documents that are relevant to the query:

$$precision = \frac{|\ relevant\ documents\ |}{|\ total\ documents\ |}$$

Commonly the precision at position n is used (P_k), in comparison to the normal formula for precision, here only the top n results of the ranked list generated by the ranking model get used in the calculation. This is also commonly written as $P@k$ in the literature.

$$P_k = \frac{|\ relevant\ documents\ in\ top\ k\ results\ |}{k}$$

The $P@k$ values are then averaged for all queries to get the mean $P@k$ value, we calculate these values up to $k = 10$ when we give our results.

Average precision can then be defined by:

$$Average\ precision = \frac{\sum_{k=1}^{n}(P(k) \times rel(k))}{|\ relevant\ documents\ |}$$

Where $rel(k)$ is an indicator function representing 1 if the document at position k is relevant and 0 if it is irrelevant, and n represents the total number of documents associated with the query.

Mean average precision is then simply the mean of the average precision for all queries.

NDCG or normalised discounted cumulative gain is an evaluation measure that is capable of using multiple graded relevance degrees . Because of this, it makes a natural choice to use as the evaluation measure for the fitness function, and is seen as the main evaluation measure of learning to rank techniques.

To describe *NDCG* we first consider *DCG* (discounted cumulative gain), which works based on the principle that highly relevant documents should be penalised more heavily the further down they are on the ranked list. The formula for *DCG*, calculated up to position k in the ranked list is then:

$$DCG_k = \sum_{i=1}^{n} \frac{2^{rel_i} - 1}{\log_2(i+1)},$$

where rel_i is the relevance degree of the document at position i.

To calculate *NDCG*, the *DCG* value must be normalised. We do this as different queries may have different numbers of documents associated with them, as such comparing the *DCG* scores alone would be bias. *NDCG* is calculated for each query by first calculating the ideal *DCG* (*IDCG*), this is the *DCG* calculated when we assume that the documents are ranked according to their ground truth labels (the ideal ranked order). The *NDCG* at position k is then:

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

The *NDCG* for each query can then be averaged to get an overall measure of the performance of the learning to rank algorithm.

We decide to use the *NDCG* as the numerical measure for the **fitness function** because being able to make use of the multiple relevance grades means its more informative than MAP or Precision, as less information is lost. To demonstrate this we consider a list of relevance degrees given in a random order $[2,4,0,0,3,1,0]$. If we then convert these to binary relevance judgements, like we would if implementing *MAP* , we obtain the list $[1,1,0,0,1,1,0]$ when using *MAP* as the fitness function we would hope that it would be able to reorder this list as close as possible to its ground truth order $[1,1,1,1,0,0,0]$. However if we then transform these back into their non-binary relevance degrees we obtain the list $[2,4,3,1,0,0,0]$ (other orders are equally possible). Therefore we find that we have not made use of the multiple graded relevance degrees and although the MAP has improved the order from a random one, it is not as good as it could be and indeed the *NDCG* score would reflect this. Less information is lost when we use the *NDCG* score as the fitness function, as it would in theory be capable of ordering them as their actual ground truth order $[4,3,2,1,0,0,0]$, where both the *MAP* scores and *NDCG* scores show good results. To summarise we hypothesise that using the most informative measure as the fitness function will maximise all other evaluation measures, whereas if we were to use a less informative measure we would not necessarily maximise the other measures.

6.4 Testing

To ensure our system is working as intended we perform tests. We note that as this project is entirely research based, we need not test as rigorously or in the same way as we would in a software based project. We include the full system testing procedure in the experimental design section of this report, however we also conducted some unit tests to ensure each part of the algorithm is performing correctly. We preform the following tests:

- We test the feature extraction system to ensure the output is of the correct form.
- We test the initialisation of the population to ensure they are generated correctly.

- We test all the crossover methods to make sure they are producing unique individuals in the form that we would expect.
- We test the mutation method, where we carefully analyse the new individual and compare its similarity to the old individual.
- We test the tournament selection method, to ensure it is choosing the best individual out of those given to be a parent.
- We test the method that creates the new population. This was particularly difficult to test and was done mainly using print statements to ensure that the new population was being changed appropriately.
- We test the evaluation method to ensure scores are numerical measures are in an appropriate range.

7 Experimental Design

To decide how well our algorithm performs we need to accurately test the system and compare the results to other data. This section describes the data we use to test the system and how we conduct the experimental procedure. We note that we have already given a large part of the experimental design in that of the evaluation system in the previous section of this document, which describes the different numerical measures that we use to assess the performance of a learning to rank algorithm.

7.1 LETOR

When deciding which data set to use for this project we considered 2 things:

- How many other algorithms can we compare our data to?
- How long does the algorithm take to train using this data set?

We find that most published papers on learning to rank use LETOR 3.0 or LETOR 4.0, these data sets are also of a good size and is manageable by our algorithm, and so using this data set makes a natural choice.

LETOR[24][29] is a collection of data sets, benchmarks and tools to be used for research on learning to rank, released by Microsoft Research Asia. Without LETOR or a similar collection, individual researchers developing learning to rank techniques would have to use their own data sets, with different queries, documents and evaluation systems. This would make it very difficult to compare results and hence make it very difficult to make significant development in the field of learning to rank. Therefore we use the LETOR collections so that we can assess the performance of our algorithm in comparison to others. There are several version of LETOR released, we choose to use the most recent *LETOR 4.0*[24].

LETOR 4.0 contains 8 data sets, with 2 data sets used for each of the following types of learning to rank algorithms; Supervised Ranking, Semi-supervised ranking, rank aggregation and direct listwise ranking. We will be using the both pieces of supervised ranking data, as this is the most common dataset used for comparison. The datasets we are using are called MQ2007 and MQ2008. These data sets have been query level normalized, so that each documents feature value ranges from 0 to 1; this is helpful for the learning function and results in faster learning. The MQ2007 data set contains roughly 1700 queries, while the MQ2008 data set contains roughly 800 queries, and in total these data sets contain 85,000 documents. Each document contains 46 document features, these features include Term frequency (how many query keywords are inside the document), IDF (inverse document frequency), document length, BM25 and PageRank, the full list of features is given in the appendix.

Each data set had been partitioned using 5-fold cross validation, with each part denoted S1, S2, S3, S4 and S5 having the same number of queries. Therefore for each fold we have 3 parts used for training, 1 part used validation

and 1 part used for testing. The table of which parts make up the training, validation and testing data for each fold is given below:

Folds	Training set	Validation set	Test set
Fold 1	{S1, S2, S3}	S4	S5
Fold 2	{S2, S3, S4}	S5	S1
Fold 3	{S3, S4, S5}	S1	S2
Fold 4	{S4, S5, S1}	S2	S3
Fold 5	{S5, S1, S2}	S3	S4

The existence of the validation data is interesting, as it allows us to validate our results by ensuring that the ranking model still performs well on unseen data sets. This ensures that our algorithm generalises well, without over-fitting the data.

In the evaluation of the system we use normalized discounted cumulative gain (*NDCG*), Precision (*P*), and Mean Average Precision (*MAP*) , as discussed in the previous section. However using these evaluation metrics causes a problem as these approaches can be implemented slightly differently, for example when calculating precision which requires binary relevance degrees, we may decide to transform relevance degrees with more than 2 grades into binary relevance degrees by letting a relevance of 0 mean irrelevant, and any relevance degree higher than that as relevant. However other people implementing precision may instead decide that a relevance degree of 0 or 1 is irrelevant and any relevance degree higher than that is relevant, both approaches wouldn't be incorrect however they would result in different results for the same data. LETOR allows us to get around this problem by providing an evaluation script written in Perl which will calculate $NDCG@k$, $MeanNDCG$, $P@k$ and MAP for us using the same method and parameters each time. This script (*Eval-Score.pl* must be run from the command line with the command:

'perl Eval-Score.pl [test file] [prediction file] [output file] [flag]'

- Here the test file is simply the test data, which contains the ground truth relevance judgments for example the file*MQ2007\Fold1\test.txt*.
- The prediction file contains the predicted relevance degree for each document in the entire set, therefore we are required to write to a file in our python script. We must be careful to ensure that the predicted relevance degrees are given in the same order as their ground truth relevance degree given in the test file.
- The output file is where all the numerical measures will be outputted.
- If the value of the flag is 1 the file will output per query results, otherwise the average evaluation results. It is these average evaluation results that we use to compare to other learning to rank systems.

We make use of Pythons read and write methods to write the predicted relevance degrees to the file, and to read from the output file. Additionally we make use of the Python package *OS* which allows us to run command line commands within Python. This allows to produce all our results without leaving the Python noteboook. We note reading and writing from the file in this way is less efficient than just using our evaluation system, hence why we don't use this for the fitness function in the learning system.

If we wish to compare our algorithm to others using LETOR there are a number of rules which we must be careful not to violate. These are:

- We must use the version of the data that has been query level normalised.

- The test set must not be used in learning or to make any decision about the parameters or structure of the ranking model.
- Results must be obtained using the evaluation script.
- The validation set can be used for model selection, but cannot be used for learning. Additionally most algorithms assessed with LETOR, that make use of the validation data use Mean Average Precision for model selection, as such LETOR highly recommend to do the same.

To make use of the validation data appropriately we decide to calculate the MAP for each ranking model in the final population on the validation data. We then decide which individual ranking model to use on the test data by choosing the individual with the highest value of $2 \times fitness + MAP$. We note that we can't simply run every ranking model in the population on the test data and use the best one as we would be breaking the second condition given above.

7.2 Experimental Plan

In this section we summarise what we want to achieve and record in the experiments that we undertake. There are two parts for this, first we want to run preliminary tests on the data, using a range of different methods and parameters, this will allow us to make difficult decision about the final version of the learning system that we have implemented. Secondly we run a variety of tests on different data sets to determine the overall performance of our algorithm.

7.2.1 Parameter Tuning

In this section we focus on tuning the parameters to be used for the final experiment. Parameter tuning is essential, as a good or bad choice of a parameters leads to a significant increase or decrease in evolutionary algorithm performance[30]. As learning to rank approaches often have very similar results a small increase in performance could lead to massive difference in terms of how it ranks compared to other algorithms. In this section we decide to conduct tests to determine on the values of the following parameters and methods: We decide on parameter tuning the following parameters:

- Which crossover method to use.
- Which value of k should be used for k-way tournament selection
- Initial Mutation rate

As mentioned in the previous section of this report, we are not sure which crossover method will perform the best, therefore we test all of the ones we have implemented, uniform crossover, single-point crossover and 2-point crossover. To see which performs the best.

For the value of k to use in k-way tournament selection we hypothesise that the lower the value of k, the longer the population will maintain its diversity and the higher the likely hood of finding the global optimum. We decide to test this hypothesis by testing for values of k in the set $\{2,3,5,10\}$.

For the initial mutation rate we will test mutation rates from the following set $\{0.01, 0.03, 0.05\}$. We are confident that mutation rates higher than this will not perform well. We hypothesise that the lower the initial mutation rate the better the algorithm will perform. This is because a lower mutation rate means it will take longer for the algorithm to converge on a solution, therefore it is less likely to miss a global optimal solution by maintaining a diverse population.

We have decided to use a population of 150 for these experiments and we decide to conduct 1500 generations. We have decided on this population size and generation number as its the largest we can practically have them without the algorithm taking too long to run, as early testing indicated that it would take exactly 24 hours to run on the larger, MQ2007 dataset. To save time for the parameter tuning section of this report we decide to only run our algorithm on Fold1 of the MQ2007 dataset. Additionally because of all the different results we need to collect we will use several computers at the same time, each running a different version of the algorithm.

For the parameter tuning experiments we will use as default single point crossover, a mutation rate of 0.03 and k=2 tournament selection. Additionally to save space in this report we will only include the following values: P_3, P_5, P_{10}, MAP, $NDCG_3$, $NDCG_5$, $NDCG_{10}$ and $NDCG_\infty$.

7.2.2 Plan Of Experiments

In each of these experiments we will calculate Precision, P_k, $k \in \{1,2,...,10\}$, Mean Average Precision, MAP, Normalised Discounted Cumulative Gain $NDCG_k$, $k \in \{1,2,..,10\}$ and Mean $NDCG$. We note that Mean $NDCG$ is equivalent to $NDCG_\infty$, where the full ranked list is considered.

After we have completed parameter tuning we will run the experiments on the MQ2007 and MQ2008 data sets, for each of the 5 sets partitioned using 5-cross fold validation. We will then average the achieved results across all 5 folds.

Running the experiments consists of the training of the learning system using the relevant piece of training data, followed by selecting the ranking model to use using the validation data, followed by using the chosen ranking model on the test data and then using the LETOR 4.0 evaluation tool to test the ranking models performance. This will give us the results of all the numerical measures described above.

7.3 Comparisons

We decide to compare our results to the current **top preforming** algorithms, however we are limited to algorithms that have been run on the LETOR 4.0 data sets. The following algorithms will be compared in this report:

- AdaRank - MAP[20] - A listwise technique which is of particular interest as like out algorithm it directly makes use of the evaluation measures used for testing, in the training method. In this case MAP is used.
- AdaRank - NDCG[20] - A listwise technique which works the same as the previous technique only NDCG is used as the evaluation measure used for training, like our algorithm. Should we find that our algorithm performs better or equal to this we would consider that an exceptional result.
- ListNet[31] - Another listwise technique which has been shown to perform very well.
- RankBoost[32] - RankBoost is considered one of the top performing pairwise approaches[33], in RankBoost a boosting approach.
- Rank SVM-Struct[34] - A very well performing pairwise approach that makes use of support vector machines, it is also shown to be a very well performing algorithm.
- RankNet[9] - A pairwise technique that makes use of neural networks, this is one of the algorithms that is currently used in industry, Bing uses an algorithm very similar to RankNet.
- LamdaMART[35] - A listwise technique which is one of the best current solutions to the ranking problem[36], winning track 1 of the Yahoo Learning To Rank Challenge[37], an online competition where ranking algorithms are compared against each other.

8 Results And Evaluation

8.1 Parameter Tuning Results

We first compare the different crossover functions that we can use:

CrossoverFunction	P_3	P_5	P_{10}	*MAP*	$NDCG_3$	$NDCG_5$	$NDCG_{10}$	$NDCG_\infty$
Uniform Crossover	0.4484	0.4298	0.3935	0.4778	0.4253	0.4395	0.4633	0.5101
Single-Point Crossover	**0.4573**	**0.4393**	0.3946	**0.4868**	**0.4439**	**0.4546**	**0.4735**	**0.5233**
Two-Point Crossover	0.4563	0.4363	**0.3973**	**0.4868**	0.4389	0.4501	0.4712	0.5200

From this we conclude that single point crossover is the best to use, and uniform crossover is significantly worse than the other two approaches. This is an interesting finding, which may also be useful for other researchers. Therefore we decide to use single point crossover for our final experiments.

Next we compare possible values of k to use in k-way tournament selection when deciding on the parents for the next generation:

k value	P_3	P_5	P_{10}	*MAP*	$NDCG_3$	$NDCG_5$	$NDCG_{10}$	$NDCG_\infty$
2	0.4573	**0.4393**	0.3946	0.4868	0.4439	**0.4546**	0.4735	**0.5233**
3	0.4593	0.4369	**0.3973**	**0.4899**	**0.4450**	0.4540	**0.4757**	0.5232
5	0.4563	0.4357	0.3958	0.4866	0.4415	0.4506	0.4733	0.5217
10	**0.4653**	0.4357	0.3958	0.4889	0.4448	0.4521	0.4734	0.5217

We hypothesised that higher values of k converge on a good solution faster, but in doing this a global optimum is more likely to be missed, hence a lower value of k should perform better on average. The data although with some amount of randomness present, does appear to support this therefore we decide to use 2-way tournament selection. Should we wish to speed up our algorithm we could increase the value of k, without a significant drop in performance.

Next we investigate different values for the initial mutation rate:

Mutation Rate	P_3	P_5	P_{10}	*MAP*	$NDCG_3$	$NDCG_5$	$NDCG_{10}$	$NDCG_\infty$
0.01	**0.4692**	**0.4423**	**0.3967**	**0.4885**	**0.4505**	**0.4605**	**0.4775**	**0.5255**
0.03	0.4573	0.4393	0.3946	0.4868	0.4439	0.4546	0.4735	0.5233
0.05	0.4415	0.4268	0.3949	0.4813	0.4275	0.4417	0.4604	0.5134

A lower initial mutation rate appears to perform best as we hypothesised, however we find that although it performs better on a lower mutation rate the extra time it takes to converge on the solution probably isn't worth the increase in performance, hence we decide to compromise on performance and time to complete and use a mutation rate of 0.03.

8.2 Official Results

In this section we give the results of our final algorithm, and compare the results to other algorithms proposed in the literature. We decide to name our algorithm *RankEvolved*.

Dataset: MQ2007 (Performance on test data)

Folds	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	*MAP*
Fold 1	0.5089	0.4851	0.4573	0.4457	0.4393	0.4296	0.4196	0.4107	0.4041	0.3946	0.4868
Fold 2	0.4572	0.4425	0.4277	0.4240	0.4183	0.4086	0.4008	0.3945	0.3910	0.3870	0.4540
Fold 3	0.4867	0.4661	0.4435	0.4233	0.4124	0.3997	0.3890	0.3809	0.3773	0.3708	0.4698
Fold 4	0.4159	0.4115	0.4031	0.3968	0.3947	0.3859	0.3835	0.3794	0.3756	0.3678	0.4395
Fold 5	0.4661	0.4631	0.4376	0.4270	0.4201	0.4105	0.4012	0.3949	0.3818	0.3758	0.4691
Mean	0.4670	0.4537	0.4338	0.4234	0.4170	0.4069	0.3988	0.3921	0.3860	0.3792	0.4638

$NDCG_1$	$NDCG_2$	$NDCG_3$	$NDCG_4$	$NDCG_5$	$NDCG_6$	$NDCG_7$	$NDCG_8$	$NDCG_9$	$NDCG_{10}$	$NDCG_\infty$
0.4415	0.4526	0.4439	0.4473	0.4546	0.4587	0.4627	0.4667	0.4708	0.4735	0.5233
0.3864	0.3904	0.3889	0.3952	0.4005	0.4055	0.4104	0.4155	0.4223	0.4292	0.4823
0.4317	0.4297	0.4335	0.4348	0.4376	0.4389	0.4397	0.4439	0.4495	0.4533	0.5114
0.3530	0.3599	0.3659	0.3688	0.3792	0.3846	0.3946	0.4029	0.4103	0.4153	0.4688
0.4031	0.4233	0.4223	0.4272	0.4315	0.4364	0.4396	0.4446	0.4466	0.4518	0.5094
0.4031	0.4112	0.4109	0.4147	0.4207	0.4248	0.4294	0.4347	0.4399	0.4446	0.4990

Dataset: MQ2008 (Performance on test data)

Folds	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	*MAP*
Fold 1	0.4231	0.3974	0.3782	0.3606	0.3436	0.3141	0.2894	0.2692	0.2528	0.2378	0.4540
Fold 2	0.3822	0.3503	0.3355	0.3185	0.3019	0.2951	0.2712	0.2524	0.2350	0.2210	0.4306
Fold 3	0.3949	0.3885	0.3631	0.3392	0.3197	0.2962	0.2821	0.2651	0.2477	0.2312	0.4442
Fold 4	0.5159	0.5032	0.4522	0.4315	0.4076	0.3832	0.3594	0.3384	0.3163	0.2975	0.5374
Fold 5	0.4841	0.4299	0.3992	0.3726	0.3439	0.3206	0.3021	0.2795	0.2647	0.2484	0.5042
Mean	0.4400	0.4139	0.3856	0.3645	0.3433	0.3218	0.3008	0.2809	0.2633	0.2472	0.4741

$NDCG_1$	$NDCG_2$	$NDCG_3$	$NDCG_4$	$NDCG_5$	$NDCG_6$	$NDCG_7$	$NDCG_8$	$NDCG_9$	$NDCG_{10}$	$NDCG_\infty$
0.3462	0.3803	0.4092	0.4338	0.4555	0.4653	0.4782	0.4226	0.2062	0.2108	0.4599
0.3100	0.3471	0.3725	0.3957	0.4128	0.4365	0.4367	0.4204	0.1659	0.1694	0.4278
0.3335	0.3907	0.4086	0.4333	0.4500	0.4640	0.4681	0.4526	0.2490	0.2512	0.4699
0.4225	0.4735	0.4857	0.5155	0.5322	0.5393	0.5462	0.4967	0.2869	0.2919	0.5441
0.4119	0.4315	0.4617	0.4859	0.5052	0.5195	0.5290	0.4919	0.2119	0.2173	0.5183
0.3648	0.4046	0.4275	0.4528	0.4711	0.4849	0.4916	0.4568	0.2240	0.2281	0.4840

We compare the scores of the numerical measures for a range of different algorithms averaged across all folds in the table below, we note we have collected this data from the LETOR 4.0 benchmark data and the learning to rank academic benchmark website[38].

Dataset: MQ2007 (Performance on test data)

Algorithm	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	MAP
AdaRankMAP	0.4392	0.4301	0.4230	0.4126	0.4054	0.3953	0.3873	0.3823	0.3782	0.3738	0.4577
AdaRankNDCG	0.4475	0.4374	0.4305	0.4169	0.4068	0.3994	0.3925	0.3868	0.3826	0.3756	0.4602
ListNet	0.4640	0.4471	0.4334	0.4248	0.4126	0.4036	0.3968	0.3911	0.3847	0.3798	0.4652
RankBoost	0.4799	0.4578	0.4440	0.4252	0.4113	0.4007	0.3950	0.3898	0.3854	0.3800	0.4624
RankSVM	0.4746	0.4996	0.4315	0.4194	0.4135	0.4048	0.3994	0.3931	0.3868	0.3833	0.4645
RankNet	0.4515	0.4303	0.4129	0.4004	0.3915	0.3863	0.3791	0.3725	0.3665	0.3604	0.4500
LambdaMART	0.4811	0.4550	0.4395	0.4255	0.4193	0.4073	0.4016	0.3960	0.3904	0.3854	0.4658
RankEvolved	0.4670	0.4537	0.4338	0.4234	0.4170	0.4069	0.3988	0.3921	0.3860	0.3792	0.4638

$NDCG_1$	$NDCG_2$	$NDCG_3$	$NDCG_4$	$NDCG_5$	$NDCG_6$	$NDCG_7$	$NDCG_8$	$NDCG_9$	$NDCG_{10}$	$NDCG_\infty$
0.3821	0.3900	0.3984	0.4024	0.4070	0.4113	0.4154	0.4217	0.4278	0.4335	0.4891
0.3876	0.3967	0.4044	0.4067	0.4102	0.4156	0.4203	0.4258	0.4319	0.4369	0.4914
0.4002	0.4063	0.4091	0.4144	0.4170	0.4229	0.4275	0.4328	0.4381	0.4440	0.4988
0.4126	0.4147	0.4185	0.4191	0.4191	0.4230	0.4278	0.4332	0.4393	0.4442	0.4995
0.4096	0.4074	0.4063	0.4084	0.4143	0.4195	0.4252	0.4306	0.4362	0.4439	0.4966
0.3893	0.3895	0.3907	0.3924	0.3966	0.4038	0.4087	0.4136	0.4192	0.4235	0.4821
0.4122	0.4085	0.4115	0.4141	0.4185	0.4215	0.4278	0.4334	0.4392	0.4447	0.4985
0.4031	0.4112	0.4109	0.4147	0.4207	0.4248	0.4294	0.4347	0.4399	0.4446	0.4990

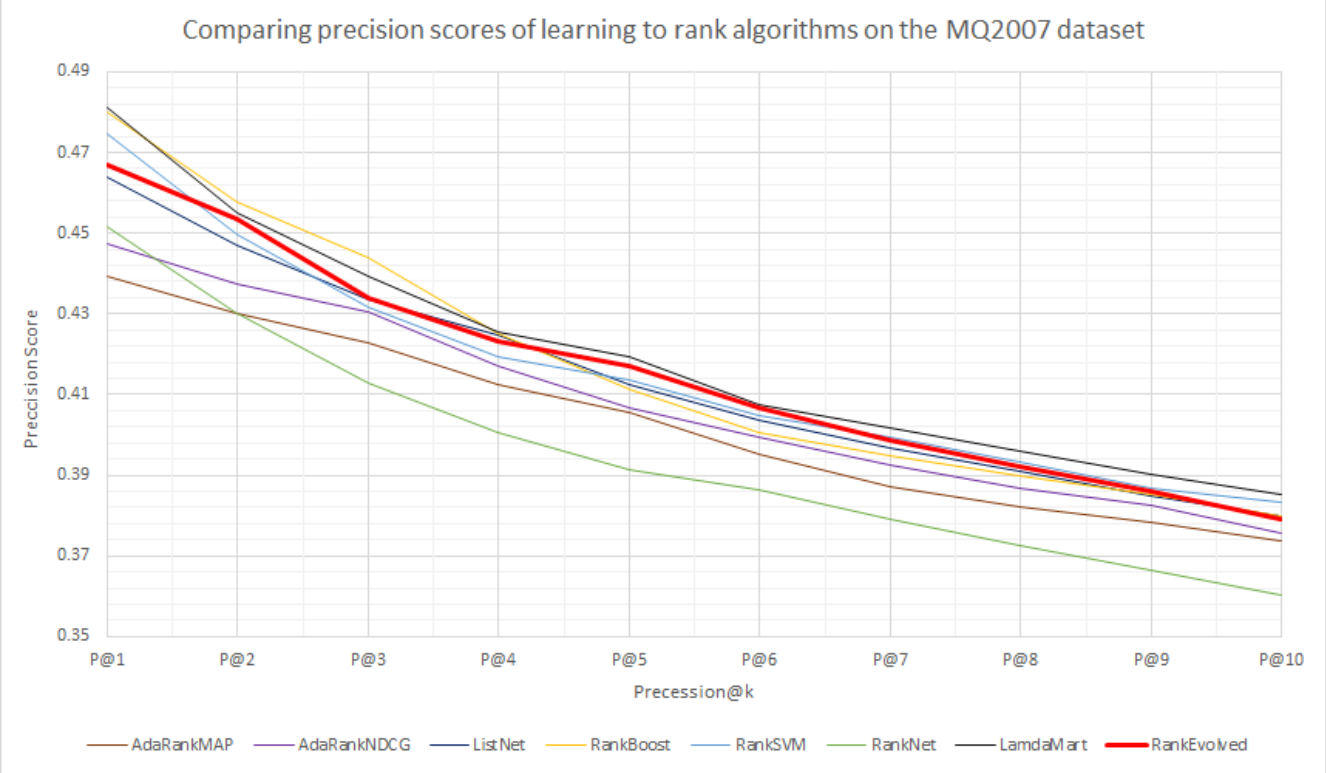


Figure Set 2

The graphs representing these tables of the MQ2007 dataset are shown in Figure Set 2, Figure Set 3 and Figure Set 4. We notice our algorithm performs particularly well with the results being competitive with the current best algorithms in learning to rank and outperforming many of them, its main competitors being LamdaMART and RankBoost. The results showing *NDCG* scores perform particularly well and is likely the result of having the fitness function directly using the *NDCG* score. Furthermore we believe that if we had used a larger population and a lower mutation rate the results obtained could have been even better than those given here, although we decided not to do this as the algorithm would have taken a very long time to train.

Dataset: MQ2008 (Performance on test data)

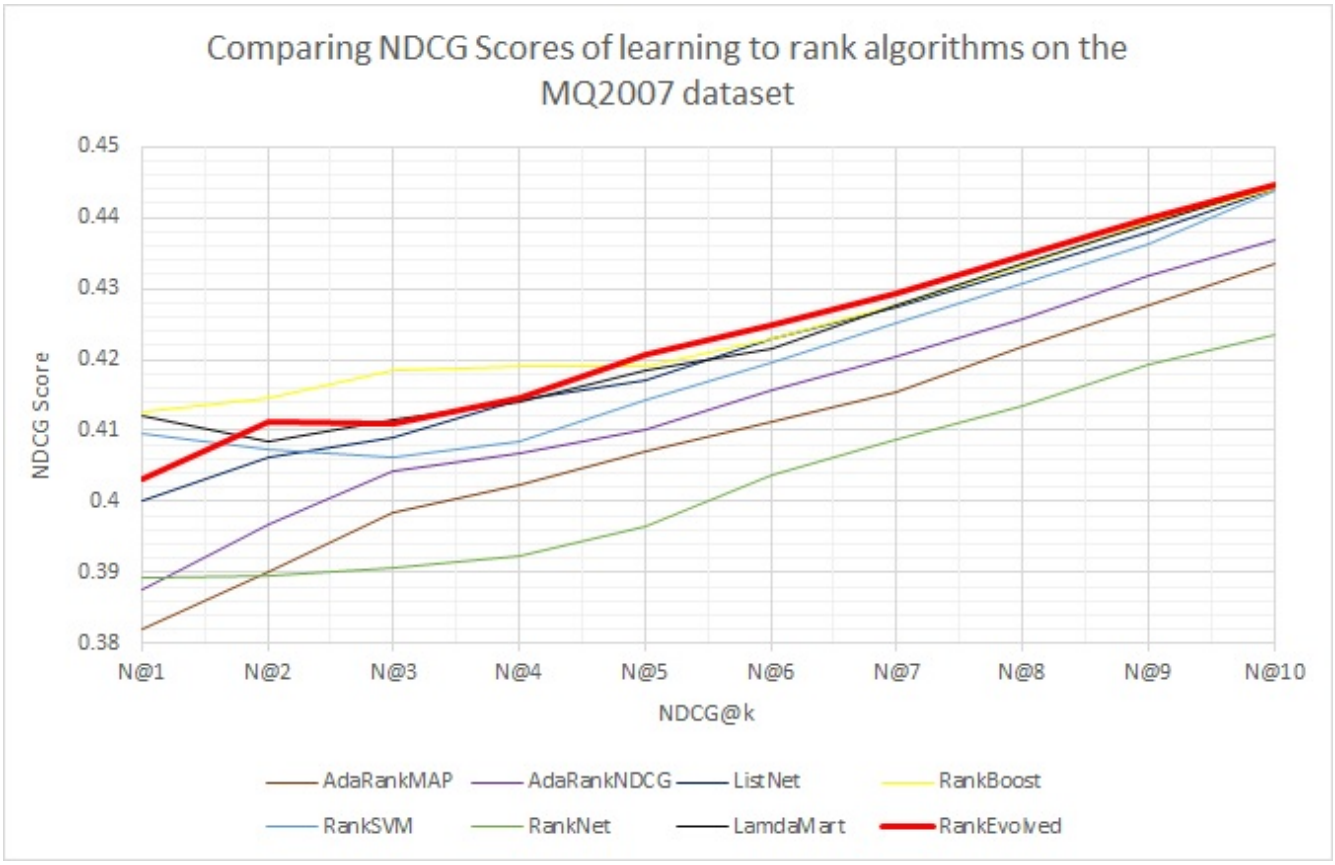
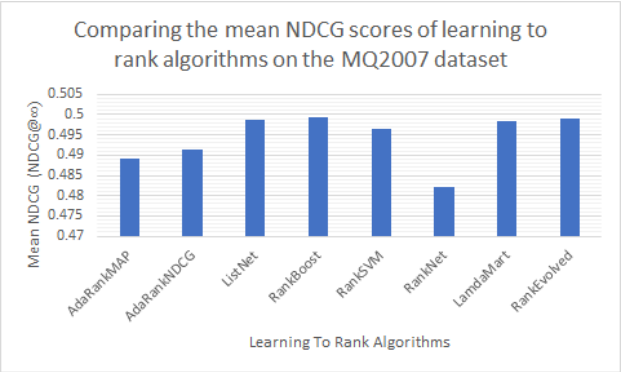


Figure Set 3

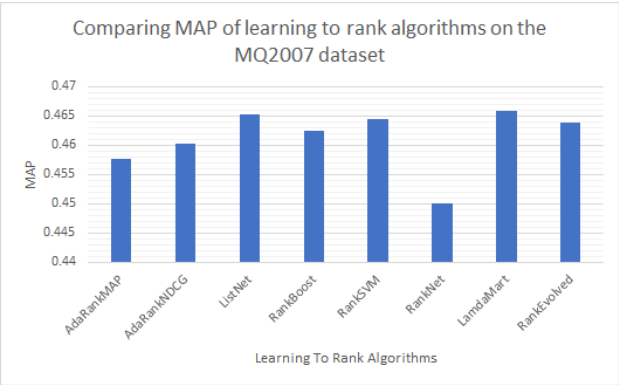
Algorithm	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	P_9	P_{10}	MAP
AdaRankMAP	0.4426	0.4165	0.3899	0.3677	0.3349	0.3216	0.2992	0.2796	0.2620	0.2454	0.4764
AdaRankNDCG	0.4515	0.4222	0.3950	0.3696	0.3454	0.3227	0.2993	0.2795	0.2618	0.2454	0.4824
ListNet	0.4451	0.4120	0.3885	0.3651	0.3426	0.3204	0.3006	0.2791	0.2627	0.2476	0.4775
RankBoost	0.4413	0.4094	0.3962	0.3689	0.3480	0.3280	0.3063	0.2855	0.2666	0.2508	0.4758
RankSVM	0.4273	0.4069	0.3903	0.3696	0.3474	0.3265	0.3021	0.2822	0.2647	0.2491	0.4696
RankNet	0.4068	0.3839	0.3622	0.3466	0.3280	0.3095	0.2888	0.2713	0.2545	0.2403	0.4514
LambdaMART	0.4489	0.4113	0.3843	0.3596	0.3405	0.3203	0.2971	0.2791	0.2600	0.2440	0.4753
RankEvolved	0.4400	0.4139	0.3856	0.3645	0.3433	0.3218	0.3008	0.2809	0.2633	0.2472	0.4741

$NDCG_1$	$NDCG_2$	$NDCG_3$	$NDCG_4$	$NDCG_5$	$NDCG_6$	$NDCG_7$	$NDCG_8$	$NDCG_9$	$NDCG_{10}$	$NDCG_\infty$
0.3754	0.4141	0.4370	0.4609	0.4794	0.4917	0.4970	0.4609	0.2254	0.2288	0.4915
0.3826	0.4211	0.4420	0.4653	0.4821	0.4948	0.4993	0.4636	0.2270	0.2307	0.4950
0.3754	0.4112	0.4324	0.4568	0.4747	0.4894	0.4978	0.4630	0.2265	0.2303	0.4914
0.3665	0.3919	0.4281	0.4487	0.4680	0.4837	4919	0.4568	0.2248	0.2289	0.4820
0.3627	0.3985	0.4286	0.4509	0.4695	0.4851	0.4905	0.4564	0.2239	0.2279	0.4832
0.3422	0.3776	0.4042	0.4281	0.4475	0.4631	0.4712	0.4379	0.2074	0.2126	0.4642
0.3462	0.3996	0.4288	0.4486	0.4689	0.4849	0.4903	0.4568	0.2230	0.2275	0.4852
0.3648	0.4046	0.4275	0.4528	0.4711	0.4849	0.4916	0.4568	0.2240	0.2281	0.4840

The figures comparing algorithm performance of the MQ2008 scores are given in Appendix B, however we do not focus on this dataset and they are added only for completeness. We note the MQ2008 dataset is much smaller



(a) Mean NDCG



(b) MAP

Figure Set 4

than MQ2007, this means that algorithms struggle to create ranking algorithms particularly for high values of k in $NDCG_k$ and P_k as such we do not include the diagrams in the main body of this report. We see that that our algorithm doesn't perform as well on this dataset when compared to the other algorithms, although this could well be the result of the randomness obtains from having a smaller dataset.

Clearly the results achieved are exceptionally good and better than expected, as part of a good research methodology we make sure to validate that our results are indeed correct by considering what could have possibly gone wrong:

- The evaluation system is giving incorrect results - This is not possible as we are using the correct LETOR 4.0 evaluation script.
- We are using the actual relevance degrees as a feature - This is definitely not the case after checking the code, just in case we additionally check the feature constants in the ranking model, if we were using the actual relevance degree as the feature it would have to be the first one and the feature constant representing the first feature is not too large. Furthermore if we were using the actual relevance degree as a feature we would expect almost perfect ranking which is not the case.

Therefore we conclude that our results obtained are correct.

We will continue our evaluation of the algorithm and over project process and approach in the critical analysis section of this report.

9 Product And Fitness For Purpose

Clearly the results and comparison to the literature show that the learning to rank algorithm that we have developed is of very high quality and provides a competitive solution to the learning to rank problem. We must give credit however to ES-Rank, a (1+1) evolutionary strategy, for developing the mutation method which plays a large part of the success of this algorithm.

Below we list the original requirements and comment on whether or not they have been in achieved and departures from these requirements are justified. We note that this is slightly difficult to do as this is a research project and hence a lot of the requirements are very broad to allow for flexibility in the design.

- **Implement a pointwise approach** - This has been achieved (Section 4).
- **Implement a pairwise approach** - This has been achieved (Section 4).

- **Implement a listwise approach** - This has been achieved (Section 4).
- **Implement a new approach based on what has been learned from the other approaches** - This has been the main focus of this report and has been achieved. Perhaps some care should be take in our definition of *new*, as genetic programming has been used before the learning system in learning to rank in *RankGP*[39] however we represent individuals in a different way (they use a binary tree representation) and use different parent selection, mutation and crossover methods, as such we have created a unique algorithm. We note that perhaps we have departed slightly from the original requirements in that we have focused more on developing this new approach than on implementing the pointwise, pairwise and listwise approaches, however once these existing approaches were implemented and it became clear that listwise was a good choice, there was little to be gained by completing a more in depth comparison.

For each of these approaches we required a feature extraction system, a learning system, a ranking system and an evaluation system. We first consider these requirements for our algorithm that uses an evolutionary algorithm for the learning system.

- **A feature extraction system, which converts the training data to a use-able form.** - This has been implemented.
- **A learning system, which should create the ranking model.** - This has been implemented and has been the main focus of this report, we have created a learning system that is capable of producing a ranking model. We should mention that the ranking model that we have created is a linear one. Although this doesn't depart from our requirement, the limitation of the ranking model should still be considered. We have made the ranking model linear as this is by far the most common model in learning to rank and implementing a non-linear ranking model can be very difficult to get right as there a wide range of non-linear functions to choose from.
- **A ranking system, which orders the documents by the predicted relevance given by the ranking model.** - Although this has been implemented, when developing the system it became apparent that it wasn't as essential or as difficult to implement as we expected as such we decided to combine it into the evaluation system.
- **An evaluation system to measure of how well the learning to rank algorithm performs.** - This has been implemented both for use as the fitness function and to assess how well the algorithm performs. Additionally when we decide to use the LETOR data sets we implement their evaluation system that makes use of their script to make sure the results can be compared fairly to others.

These requirements have also been implemented for the pointwise, pairwise and listwise approaches that we disccussed in section 4, however we have not focused on them for the reasons already discussed.
We also consider the research problem:
'How do we improve on the ranking accuracy of learning to rank approaches?'

We believe that we have answered this question as we have found that a good way to improve on the ranking accuracy is with listwise approaches and we have demonstrated one such listwise approach that does exactly that by using a evolutionary algorithm as the learning system. Although this doesn't necessarily mean that there doesn't exist other ways of improving the ranking accuracy, leaving room for future research.

10 Critical Analysis

In this section we continue our evolution of our final implemented learning system, as well as analysing the project as a whole.

10.1 Limitations

Because of the chosen fitness function, the evolutionary algorithm is very slow to train compared to other methods. This is fine for the small amount of data we are using currently, however if we were to increase the size of our training data significantly, our algorithm would be too slow to be trained without the use of a supercomputer. To summarise our algorithm performs extremely well, but with its current settings and with a large population, it comes at a great computational cost.

Additionally we have mainly worked on a learning system that creates a **linear** ranking model, clearly this does produce a good solution to the problem, but a non-linear solution would in theory perform better. This can be demonstrated by considering some specific cases. For example several document features that we use are related to the number of query keywords in the document. When using the linear approach a high score in these features is a good indicator that the document is relevant. However this is not necessarily the case, for example you could simply have a document that repeats common query keywords over and over again without any actual useful content. In this case the document would have a ground truth relevance of 0, but be classified by the ranking algorithm as highly relevant. Instead a better solution would be to combine features, for example we might combine the number of query keywords in the document, with the number of inlinks the document has (number of hyperlinks leading to that document). This would be represented as $\alpha_1 * \alpha_2$ and would ensure that these spam documents, which wont have many inlinks will not produce a high score. There are likely many more of these cases, with a more complicated non-linear relationship than the one we suggest here.

10.2 Project Process And Approach

Clearly the results obtained in the project were very successful however we could have made improvements on the process and approaches that we used in the project.

Firstly I think too much time was spent attempting to code existing learning to rank techniques, this proved to be extremely difficult and time consuming. However this was still somewhat useful as it really helped in the understanding of learning to rank, which can make a confusing first read.

Secondly related to the first point, it would have been preferred to have simply begun investigating evolutionary algorithms acting as the learning system for learning to rank from the start of this project, rather than implementing and investigating several different ares, such as how different loss functions effect performance.This meant a lot of additional research had to be done on evolutionary algorithms and genetic programming, leaving only a few months to construct and run the tests on the final algorithm. Although without the implementing these different approaches, it is unlikely that I would have decided to use evolutionary algorithms as the learning system.

Thirdly I think it would have been been preferred to have compared our results against other datasets but because of the training time of the algorithm and the parameter tuning that would have to be done for each dataset this wasn't possible.

Additionally it would have been good to experiment more with different variations of the mutation method, and different fitness functions but we were limited by the time constraints due to the training times of the algorithm.

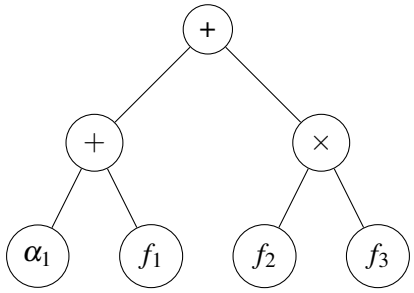
11 Future Work

To conclude this work we have seen that listwise approaches provides very good results of the ranking task, and we believe this approach is probably better than the pointwise and pairwise approaches. We have also designed and implemented a new algorithm,*RankEvolved*, which uses an evolutionary algorithm as the learning system. We have seen that this design is a very competitive with the best existing learning to rank algorithms which is a huge success. However we find drawbacks in the method, the main one being the time it takes the algorithm to train which may make it unusable for some purposes. However learning systems of this form are clearly worth investigation further.

Future work would include further experimentation of LETOR 3.0 and some of the larger data sets such as the Yahoo Learning to rank challenge. This would allow us to compare our results against a greater number and range of algorithms. Additionally by using larger datasets with more document features, far more insight will be given about different algorithms performance. For example we see how similar the ranking algorithms perform in the smaller MQ2008 dataset when compared to the MQ2007 dataset.

It would also be beneficial to experiment with the fitness function, to see if a less computationally expensive method can be used. without sacrificing too much in terms of performance, if such a solution can be found the viability of genetic programming for learning to rank would be greatly increased. We could go further with this idea by creating an algorithm that focuses entirely on the speed of convergence to a solution and not caring as much about performance, for example we could increase the value of k in tournament selection and slightly increase the mutation rate.

Future work could also on focus on implementing a learning system that generates a non-linear ranking model. One way of doing this would be to represent the non-linear ranking models with a binary relevance tree, where leaf nodes can be constants or feature variables and internal nodes are chosen from a list of potential operators, for example $\{x, +\}$ although more operators should be used for better performance. A small example tree structure representing the non-linear function $((\alpha_1 + f_1) + (f_2 \times f_3))$ is given below:



For such an individual the mutation method would involve changing around nodes in some way.

References

[1] H. Li, "A short introduction to learning to rank," *IEICE TRANSACTIONS on Information and Systems*, vol. 94, no. 10, pp. 1854–1862, 2011.

[2] H. Li, "Learning to rank for information retrieval and natural language processing, second edition," *Synthesis Lectures on Human Language Technologies*, vol. 7, no. 3, pp. 1–121, 2014.

[3] M. de Kunder, "The size of the world wide web (the internet)." https://www.worldwidewebsize.com/, 2018.

[4] "Google search statistics." http://www.internetlivestats.com/google-search-statistics/, 2018.

[5] R. Fishkin, "How search engines operate." https://moz.com/beginners-guide-to-seo/how-search-engines-operate, 2018.

[6] S. Robertson, H. Zaragoza, *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[7] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web.," tech. rep., Stanford InfoLab, 1999.

[8] A. Burkov, *The hundred-page machine learning book*. Andriy Burkov, 2019.

[9] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. N. Hullender, "Learning to rank using gradient descent," in *Proceedings of the 22nd International Conference on Machine learning (ICML-05)*, pp. 89–96, 2005.

[10] R. Baeza-Yates, B. Ribeiro-Neto, *et al.*, *Modern information retrieval*, vol. 463. ACM press New York, 1999.

[11] K. Järvelin and J. Kekäläinen, "Ir evaluation methods for retrieving highly relevant documents," in *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 41–48, ACM, 2000.

[12] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[13] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online; accessed 1/12/2018].

[14] T. Oliphant, "NumPy: A guide to NumPy." USA: Trelgol Publishing, 2006–. [Online; accessed 1/12/2018].

[15] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[16] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[17] T. Qin and T. Liu, "Introducing LETOR 4.0 datasets," *CoRR*, vol. abs/1306.2597, 2013.

[18] T.-Y. Liu *et al.*, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009.

[19] V. Dang, "Ranklib-a library of ranking algorithms," 2013.

[20] J. Xu and H. Li, "Adarank: a boosting algorithm for information retrieval," in *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 391–398, ACM, 2007.

[21] S. E. Robertson and S. Walker, "Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval," in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 232–241, Springer-Verlag New York, Inc., 1994.

[22] S. E. Robertson and K. S. Jones, "Relevance weighting of search terms," *Journal of the American Society for Information science*, vol. 27, no. 3, pp. 129–146, 1976.

[23] S. Robertson, H. Zaragoza, *et al.*, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009.

[24] T. Qin and T.-Y. Liu, "Introducing letor 4.0 datasets," *arXiv preprint arXiv:1306.2597*, 2013.

[25] B. L. Miller, D. E. Goldberg, *et al.*, "Genetic algorithms, tournament selection, and the effects of noise," *Complex systems*, vol. 9, no. 3, pp. 193–212, 1995.

[26] N. M. Razali, J. Geraghty, *et al.*, "Genetic algorithm performance with different selection strategies in solving tsp," in *Proceedings of the world congress on engineering*, vol. 2, pp. 1–6, International Association of Engineers Hong Kong, 2011.

[27] r0land, "Singlepointcrossover," 12 2013. License: Creative Commons Attribution-Share Alike 3.0 Unported.

[28] O. A. S. Ibrahim and D. Landa-Silva, "Es-rank: evolution strategy learning to rank approach," in *Proceedings of the Symposium on Applied Computing*, pp. 944–950, ACM, 2017.

[29] T. Qin, T.-Y. Liu, J. Xu, and H. Li, "Letor: A benchmark collection for research on learning to rank for information retrieval," *Information Retrieval*, vol. 13, no. 4, pp. 346–374, 2010.

[30] A. E. Eiben and S. K. Smit, "Evolutionary algorithm parameters and methods to tune them," in *Autonomous search*, pp. 15–36, Springer, 2011.

[31] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, ACM, 2007.

[32] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *J. Mach. Learn. Res.*, vol. 4, pp. 933–969, 2003.

[33] R. Busa-Fekete, G. Szarvas, T. Elteto, and B. Kégl, "An apple-to-apple comparison of learning-to-rank algorithms in terms of normalized discounted cumulative gain," in *ECAI 2012-20th European Conference on Artificial Intelligence: Preference Learning: Problems and Applications in AI Workshop*, vol. 242, Ios Press, 2012.

[34] T. Joachims, "Training linear svms in linear time," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 217–226, ACM, 2006.

[35] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao, "Adapting boosting for information retrieval measures," *Information Retrieval*, vol. 13, no. 3, pp. 254–270, 2010.

[36] C. J. Burges, "From ranknet to lambdarank to lambdamart: An overview," *Learning*, vol. 11, no. 23-581, p. 81, 2010.

[37] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview," in *Proceedings of the Learning to Rank Challenge*, pp. 1–24, 2011.

[38] "Academic benchmark learning to rank." http://www.bigdatalab.ac.cn/benchmark/bm/Domain?domain=Learning%20to%20Rank, 2015.

[39] J.-Y. Yeh, J.-Y. Lin, H.-R. Ke, and W.-P. Yang, "Learning to rank for information retrieval using genetic programming," in *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval (LR4IR 2007)*, 2007.

Appendix A Document Features

Column in Output	Description
1	TF(Term frequency) of body
2	TF of anchor
3	TF of title
4	TF of URL
5	TF of whole document
6	IDF(Inverse document frequency) of body
7	IDF of anchor
8	IDF of title
9	IDF of URL
10	IDF of whole document
11	TF*IDF of body
12	TF*IDF of anchor
13	TF*IDF of title
14	TF*IDF of URL
15	TF*IDF of whole document
16	DL(Document length) of body
17	DL of anchor
18	DL of title
19	DL of URL
20	DL of whole document
21	BM25 of body
22	BM25 of anchor
23	BM25 of title
24	BM25 of URL
25	BM25 of whole document
26	LMIR.ABS of body
27	LMIR.ABS of anchor
28	LMIR.ABS of title
29	LMIR.ABS of URL
30	LMIR.ABS of whole document
31	LMIR.DIR of body
32	LMIR.DIR of anchor
33	LMIR.DIR of title
34	LMIR.DIR of URL
35	LMIR.DIR of whole document
36	LMIR.JM of body
37	LMIR.JM of anchor
38	LMIR.JM of title
39	LMIR.JM of URL
40	LMIR.JM of whole document
41	PageRank
42	Inlink number
43	Outlink number
44	Number of slash in URL
45	Length of URL
46	Number of child page

Figure Set 5: Document features for the LETOR 4.0 datasets

Appendix B MQ2008Figures

We have included figures of the MQ2008 data here, we do not focus on this dataset and the graphs shown here are not necessary for this report, as such they are added only for completeness.

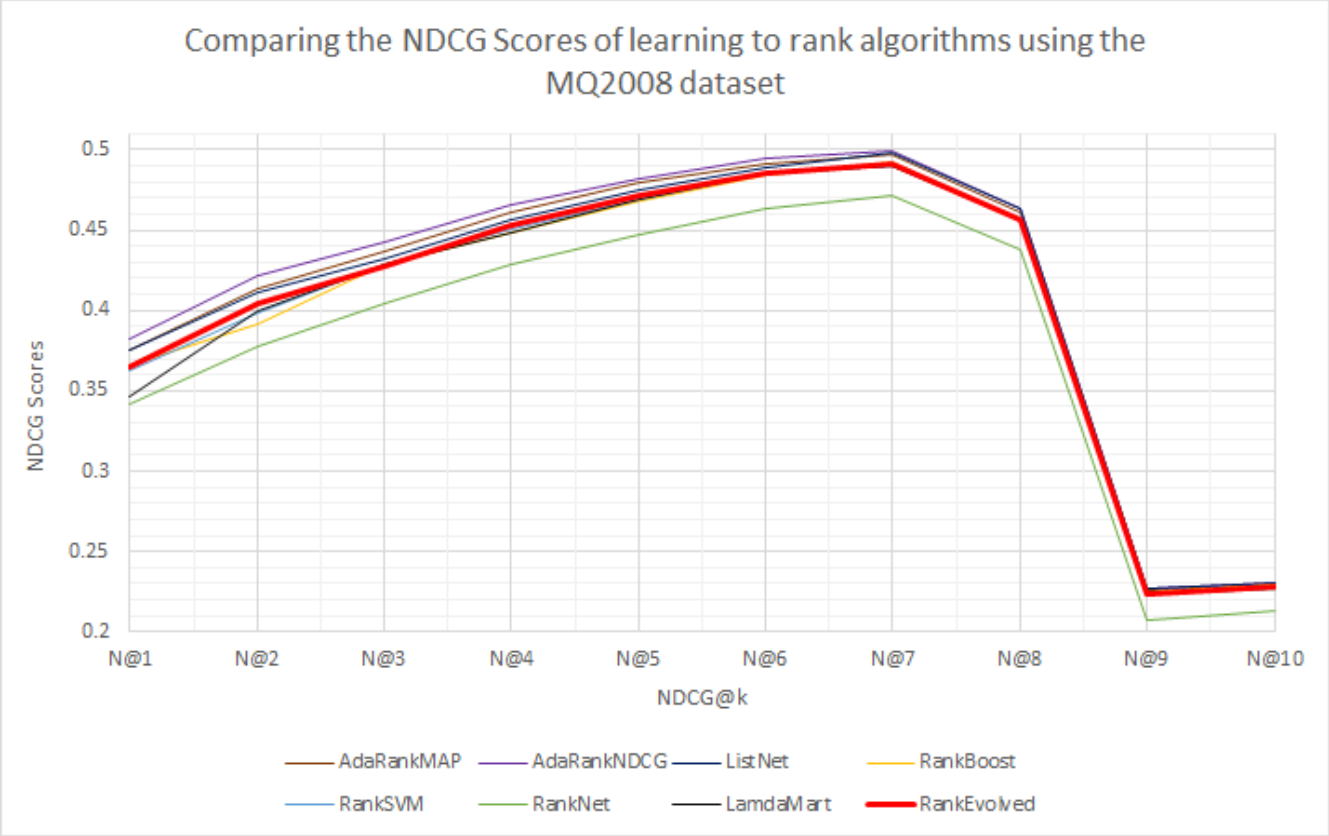


Figure Set 6

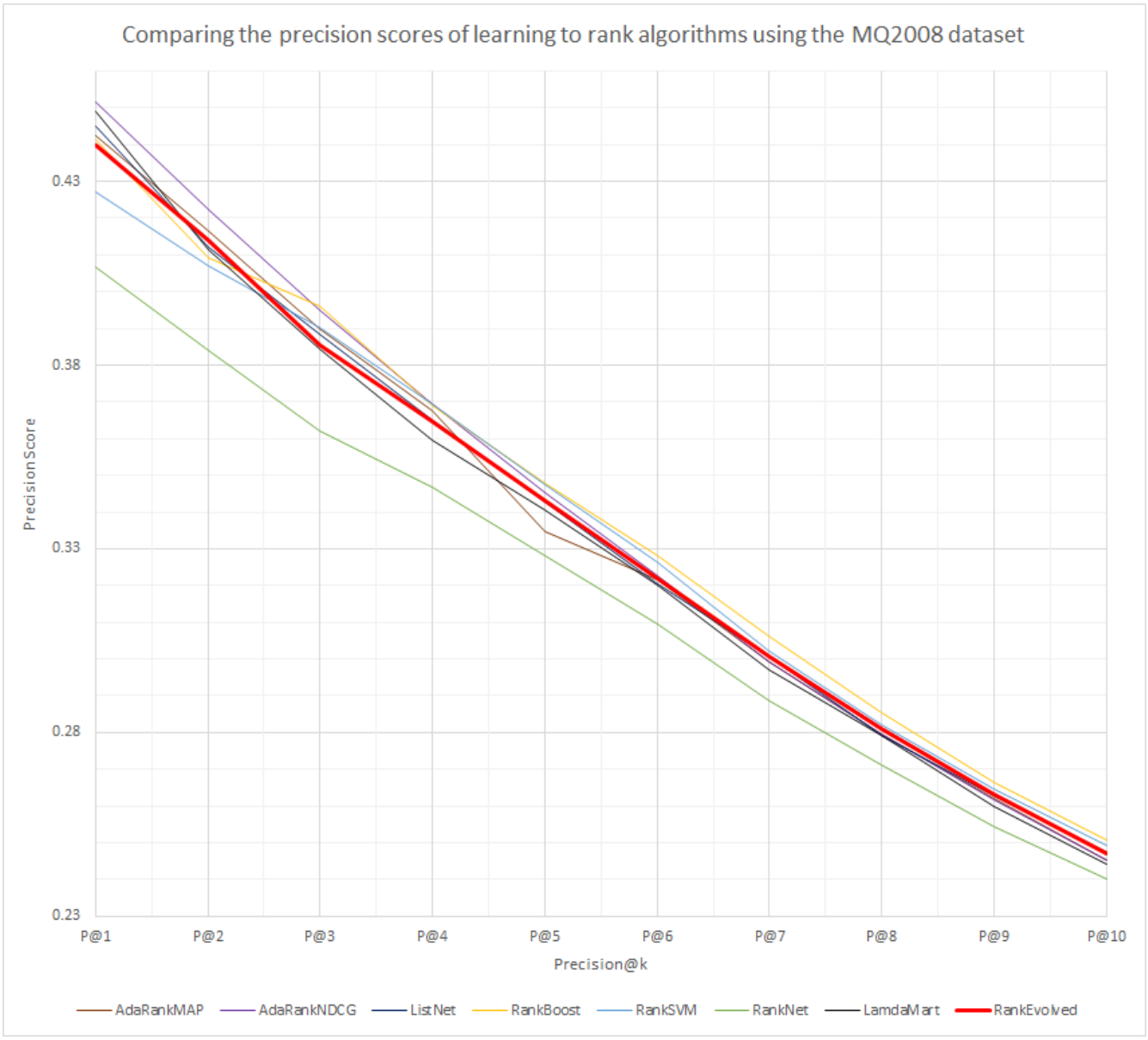

Comparing the precision scores of learning to rank algorithms using the MQ2008 dataset
Precision Score
0.43
0.38
0.33
0.28
0.23
P@1
P@2
P@3
P@4
P@5
P@6
P@7
P@8
P@9
P@10
Precision@k
AdaRankMAP
AdaRankNDCG
ListNet
RankBoost
RankSVM
RankNet
LamdaMart
RankEvolved

Figure Set 7

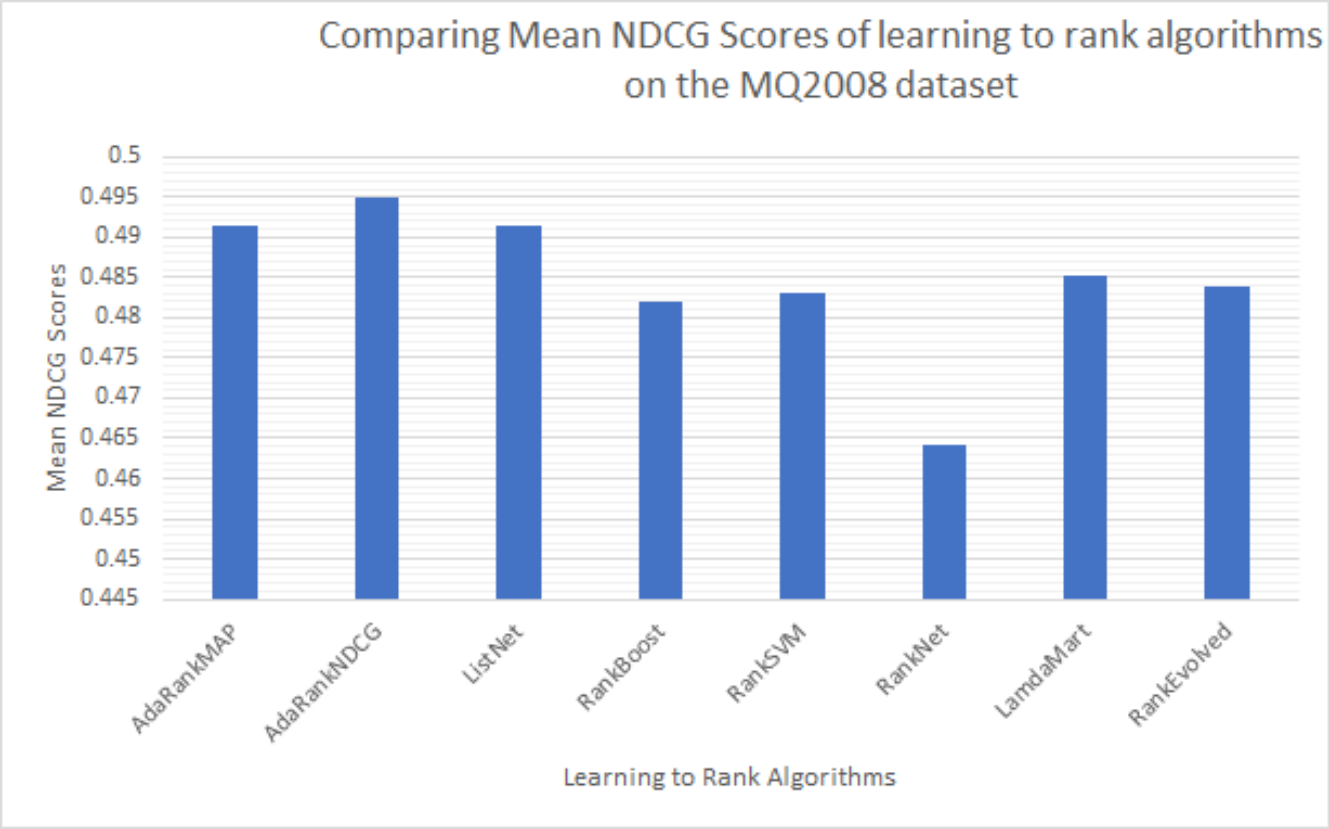

Comparing Mean NDCG Scores of learning to rank algorithms on the MQ2008 dataset
Mean NDCG Scores
0.5
0.495
0.49
0.485
0.48
0.475
0.47
0.465
0.46
0.455
0.45
0.445
AdaRankMAP
AdaRankNDCG
ListNet
RankBoost
RankSVM
RankNet
LamdaMart
RankEvolved
Learning to Rank Algorithms

Figure Set 8

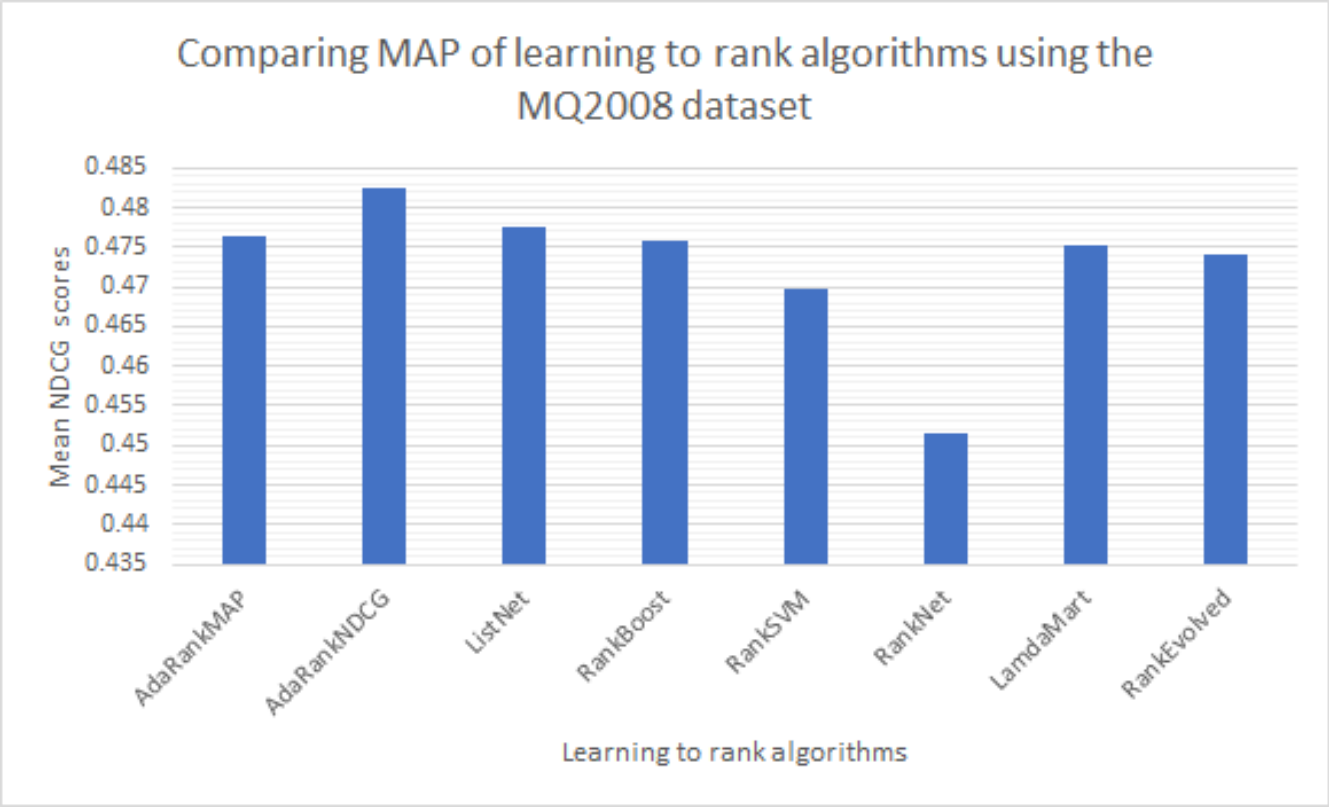

Comparing MAP of learning to rank algorithms using the MQ2008 dataset
Mean NDCG scores
0.485
0.48
0.475
0.47
0.465
0.46
0.455
0.45
0.445
0.44
0.435
AdaRankMAP
AdaRankNDCG
ListNet
RankBoost
RankSVM
RankNet
LamdaMart
RankEvolved
Learning to rank algorithms

Figure Set 9