Name: Stylianos Rousoglou
Department: Computer Science

Information Retrieval and Data Mining: Group 09 Report

Swakeert Jain
MSc WSBDA
swakeert.jain.16@ucl.ac.uk

Stylianos Rousoglou
Affiliate Student, Computer Science, University College London
stylianos.rousoglou.16@ucl.ac.uk

Gurpreet Singh
MSc WSBDA
gurpreet.singh.16@ucl.ac.uk

Tiegsti Solomon
MSc WSBDA
ucabths@ucl.ac.ul

ABSTRACT

Learning to rank for Information Retrieval (IR) is a task to automatically construct a ranking model using training data, such that the model can sort new objects according to their degrees of relevance, preference, or importance. There are a huge number of IR problems that by nature are ranking problems, and can therefore be enhanced using Learning-to-rank techniques. In this paper, the three of the best algorithms for LTR are implemented and their performance compared by various metrics to determine each's performance against the others. Analyses is performed on each of the approaches and advantages and disadvantages are discussed for each of the algorithms. A machine learning approach to learning to rank trains a model to optimize a target evaluation measure with respect to training data. The paper is concerned with learning to rank, which is to construct a model or a function for ranking objects. Learning to rank is useful for document retrieval, collaborative filtering, and many other applications. To assess these techniques, we work with the MSRDataset, provided by Microsoft.

Keywords

Ground Truth (GT), Ranking Function (RF), Ideal Discounted Cumulative Gain (IDCG), Normalised Discounted Cumulative Gain (NDCG), Discounted Cumulative Gain (DCG), Mean Average Precision (MAP)

1. INTRODUCTION

The internet has grown extremely rapidly in size over the past few decades. The amount of indexed web pages has crossed 50 billion [1] today and even more unknown and unlisted websites are present outside the documented internet. Out of these billions of pages, there are several thousand sets of websites that provide the same or similar kind of information to the user. For example, if we think about checking the news or sports scores, we can name 5 to 10 websites off the top of our heads that provide this information. To find any other information or gain any knowledge from the web, we need to have a mechanism to tell us which website to look at for a kind of information. This is where search engines come in.

A search engine is a software application that crawls the internet and indexes the data that it comes across. Then, when a user wishes to search for something, they will feed their query into the search engine and it will return a list of pages that contain that information. A search engine's core functions can be separated into two broad parts; Crawling and Ranking. The first part deals with gathering and storing all the data and metadata about all the pages that it comes across and the second part deals with how to Rank them or in which order to display them in terms of relevance when a user types in a query. This can, together, be referred to as Information Retrieval.

Ranking is a major task that requires a lot of development and is something that might never be perfected. The basic concept being that one web page might be more relevant to one user as compared to another one. Therefore, today major search engines are not only considering the content of the website when returning the results, but are also considering the user's personal data to better understand what kind of pages they are more likely to visit and thereby generate results unique to the user. Initially ranking involved displaying web pages which contain the queried keyword most number of times, and similar linear approaches like document length and inverse document frequency. This led to the search results becoming full of spam as content creators would flood their pages with keywords to attract visitors.

The search engines were then enhanced such that they also took into consideration the reputation of the website, in which it measures the page rank by considering in-links and out-links for the page, checking frequency of search term in anchor texts and metadata, URL properties, relevant images, etc. The reputation factor was further enhanced by taking into consideration how well the websites that link into and out of this page rank. This resulted in all the websites paying more attention towards maintaining its page rank and therefore improving the quality of content on the internet as a whole. It is said that major search engines like Google and Bing now take more than 200 factors into consideration while ranking the pages.

Ranking is the central problem in Information Retrieval. Many Information Retrieval problems are by nature ranking problems, such as document retrieval, collaborative filtering, key term extraction, definition finding, important email routing, sentiment analysis, product rating, and anti-web spam. In this paper, we will mainly discuss document retrieval. It is to be noted that document retrieval is not a narrow task. Web pages, academic papers, books, emails, and news articles are just a few examples of documents. To tackle the problem of document retrieval, many heuristic ranking models have been proposed and used in IR literature. Recently, given the amount of potential training data available, it has become possible to leverage Machine Learn-

[1] http://www.worldwidewebsize.com

ing technologies to build effective ranking models. Specifically, we call those methods that learn how to combine predefined features for ranking by means of discriminative learning-to-rank methods.

Learning to rank, when applied to document retrieval, is a task as follows. Assume that there is a collection of documents. In retrieval (i.e., ranking), given a query, the ranking function assigns a score to each document, and ranks the documents in descending order of the scores. The ranking order represents the relevance of documents with respect to the query. In learning, a number of queries are provided; each query is associated with a perfect ranking list of documents; a ranking function is then created using the training data, such that the model can precisely predict the ranking lists in the training data. In recent years, learning to rank has become a very hot research direction in IR, and a large number of learning-to-rank algorithms have been proposed, out of which we discuss and compare three in our paper.

There are a lot of datasets available to implement, test and improve ranking upon. The six[2] most common ones are: LETOR 3.0 - Gov; LETOR 3.0 - Ohsumed; LETOR 4.0; Yandex; Yahoo!; Microsoft. Out of these, the Yahoo! dataset contains the most number of queries, exceeding Microsoft by approximately 5k, however it is the Microsoft dataset that has the most number of docs, exceeding every other by atleast 2,900k containing 3,771k overall. Microsoft dataset also stands behind both Yahoo! and Yandex in terms of number of features with Yahoo! having an outstanding 700 features. Both the Yahoo! and Microsoft one's were published in 2010, while others are from 2008 and 09. In all the datasets, the data consists of three types, namely, query, document and grade, where the query and document features are combined. Here, the grade indicated the degree of relevance to the document to its corresponding query. These are labelled by human editors.

For our implementation, we choose the Microsoft dataset[3] mainly because it has already been partitioned into five different datasets with almost the same number of queries which provides us the possibility of a fivefold cross validation which leaves a very small scope for overfitting, which is a very common occurrence in any Machine Learning model. In each fold, we use three parts for training, one part for validation, and the remaining part for test. The validation set is used to tune the hyper parameters of the learning algorithms while the test set is used to evaluate the performance of the learned ranking models. The metrics of the dataset are further discussed in the dataset statistics section of this paper.

2. RELATED WORK

The extensive literature review performed covers a range of academic publications and articles. The scientific papers studied present and evaluate different learning to rank algorithms, and propose a variety of evaluation metrics that can be used to optimize and assess learning.

Generally, given a large volume of training data, machine learning problems are concerned with producing a model with good enough predictive capabilities on an instance-to-instance basis to consistently estimate, as closely and precisely as possible, the value of some unknown variable in real-life scenarios. Unlike traditional machine learning problems, the nature of Learning to Rank places emphasis on the model's capability to evaluate query-document instances *in relation to one another*. In other words, in the case of learning to rank, the relative scoring of different instances by our predictor model is more important than the absolute relevance score attributed to any given instance. For example, in the case of search engine ranking, a typical application for learning to rank algorithms, the user is concerned with the relevance of top results chosen among millions of documents, rather than the underlying relevance score calculated by the algorithm to achieve the particular ranking and its implications about a particular instance.

Irrespective of the problem being solved, all machine learning solutions require evaluation metrics that can be continuously calculated and used to train and optimize the ML model. Good evaluation metrics are critical for the success of a machine learning model optimizing on those metrics, and there are several different ones introduced in literature, such as MAP, MRR, and DCG, to mention a few. Mean Average Precision, or MAP, computes the precision of the ranking of each relevant document given a user query and averages it over all relevant documents. Mean Reciprocal Rank (MRR), meanwhile, computes the average Reciprocal Rank rank over all user queries, where the RR of a single query is the reciprocal of the rank position of the highest ranking relevant document in the ranked list.

Though MAP and MRR are commonly used in IR to estimate relevance, they are *binary* measures, i.e. they treat all documents as either relevant or not. Therefore, the need for a more complex measure that takes into account multilevel relevance arises. Discounted Cumulative Gain (DSG), perhaps the most widely used relevance measure in IR literature, iterates through the ranked list and penalizes highly relevant documents appearing lower in the list than less relevant ones with values proportional to their relevance level. The penalty value is discounted using each document's rank in the final list. DSG will be used later in this paper to evaluate the performance of our own Learning to Rank runs.

Different ranking models have been developed in Learning to Rank literature. All models can be roughly classified as either query-dependent or query-independent models, based on the techniques employed in retrieving relevant documents given a user query. In the case of query-dependent approaches, documents are retrieved based on occurrences of query terms. Query-dependent approaches vary from the Boolean model, whereby documents are either relevant or not, to more sophisticated approaches such as the Vector Space Model (VSM), that employ techniques such as calculating term frequency and inverse document frequency and using those to weight query terms differently. On the other hand, query-independent models rely exclusively on the properties of a document when deciding where the document should rank. Query-independent models are especially applicable to web search tasks, where many reliable quality metrics are available and easily recomputable for any url. For instance, the number of inlinks, outlinks, references, etc. are all indicators of the reliability/popularity of online documents that can be used to rank urls in query-independent ranking solutions.

Apart from different document retrieval approaches, there are also distinct flavors of learning to rank algorithms. Three

[2]https://moodle.ucl.ac.uk/mod/resource/view.php?id=2776065, slide 7

[3]https://www.microsoft.com/en-us/research/project/mslr

widely known LTR schemes are the pointwise, the pairwise, and the listwise approaches. While the former is on the traditional machine learning end of the spectrum, the latter two reflect the nature of learning to rank as a task concerned with relative, rather than absolute, scoring. Although the latter sections of this report treat and implement the learning to rank task as a classification problem, which as will soon be clear is considered a pointwise solution, all three approaches will be explained, compared and contrasted below.

Pointwise approaches to learning to rank are concerned with one instance/document at a time. Their input space is a single feature vector representing one document, while their output space consists of numerical relevance predictions for individual documents. Pointwise solutions attempt to make numerical predictions that reflect a document's degree of relevance to the query, without considering any other documents in the process. The loss function optimizes relevance predictions for single documents against the ground truth. Relevance scores are calculated for every instance, and the ranking is produced simply by sorting the list of documents by relevance score. Though the simplest strategy to implement, since it applies widely used regression or classification techniques to solve the problem, pointwise ranking performs the least well in practice, mainly because it considers documents independently and not relative to each other.

Unlike pointwise approaches, pairwise solutions do not attribute relevance scores to documents independently. Rather than looking at single document instances, pairwise approaches are concerned with the relative ranking of *pairs* of documents. Their input space consists of pairs of documents, while the output spaces is encoded into pairwise preferences between document pairs, which take values from the set $-1, 1$. Given a query and two documents associated with it, a pairwise solution is trained to decide on the correct *relative* position of the two documents in a potential ranking, essentially ruling on which one is more relevant to the given query. In this approach, the problem can be viewed as a classification task on document pairs. The natural goal for a pairwise approach is then to produce a final ranking that *minimizes* the number of inversions, i.e. the number of documents pairs found in the wrong order. In the limiting case in which all document pairs are correctly classified, all documents will be ranked correctly as well. Of course, classification in this context is different than in the pointwise approach context, as it is repeatedly applied to pairs of documents rather that the entirety of the document set.

Many famous ranking algorithms fall under the umbrella of pairwise approaches and are used to date in search engines, recommendation systems, etc. RankNet, LambdaRank and LambdaMART are all pairwise solutions, just to mention a few. Many new pairwise approaches developed over time and proven to improve performance according to some standardized evaluation metric have come about as combinations of earlier solutions and new insights, building on previous models but improving the techniques involved in order to boost performance. In fact, the three algorithms mentioned above are all derived from the first one, RankNet.

In the case of RankNet, the underlying classification model can be any model whose output is a differentiable function of its parameters (weights). After partitioning the training dataset by query, input feature vectors (representing documents) are mapped to score values. For a given query, those scores are mapped to a learned probability that one document should have higher rank than the other, using a sigmoid function (to restrict output to the range $[0, 1]$). In the case of neural networks being used as the underlying model, the sigmoid function mapping is particularly useful, as it has been shown to produce good probability estimates. The loss function is then applied, penalizing the difference between the output and the desired probability estimates. Stochastic Gradient Descent (SGD) is then used to reduce cost, repeatedly updating the weights of the model based on the gradient of the cost function.

Although RankNet performs reasonably well on optimizing pairwise errors, LambdaRank was introduced as an improvement over the RankNet algorithm. The fundamental insight that led to LambdaRank being introduced was the observation that gradients need not be derived from the cost function, but rather can be directly calculated and used, bypassing the difficulties introduced by flat or discontinuous functions. Burgess et. al. presented this idea and suggested that the gradients of the cost, symbolized by the Greek letter lamdba (İż), can be thought of as forces acting on documents in the list, moving them up or down in the ranked list until equilibrium. In LambdaRank, the gradients of the costs with respect to model costs are repeatedly accumulated for every document and summed to update the weights on each iteration of the algorithm. The fact that back-propagation was no longer necessary for learning, as was the case in RankNet training using neural networks, LambdaRank proved much faster in practice than its predecessor.

Finally, LambdaMART is the boosted tree version of LambdaRank, combining the existing LamdaRank model and the MART class of algorithms. The output of MART algorithms, which utilize boosted trees, is a linear combination of the outputs of a set of regression trees.[4]. The idea behind boosted trees is the following. Data is thought of as residing on the root node of a tree. Given a feature vector and a specific feature from the vector, we loop through the training data and compute a threshhold value such that vectors with a higher value for the particular feature fall to the right child, while all others fall to the left child. The threshold is calculated such that the combined least squares error from both subtrees is minimized. Therefore, there is a fundamental difference in the way LambdaMART updates its weights. Whereas LambdaRank updates all parameters after each query is examined, LambdaMART updates its weights gradually. Specifically, only the split values for the current tree nodes are calculated at each node.

The advantages of LambdaMART seem to be agreed upon by the literature studied. The speed-up observed in the case of LambdaMART appears to be significant and can be attributed to the boosted trees model being utilized instead of the neural net equivalent, which requires more time to be trained and optimized. On top of that, boosted tree ensembles are supposed to be especially well suited for multiclass classifiers, hence the improved performance of LambdaMART over both LambdaRank and RankNet (LambdaMART won the 2010 Yahoo Learning to Rank challenge).

The final approach to learning to rank is known as the listwise approach. The input space of listwise algorithms consists of the entire list of documents associated with a

[4]Burgess, 11

given query, rendering the problem significantly more complex than predicting relevance scores for individual documents (pointwise) or deciding between pairs of documents (pairwise). Depending on the flavor of the listwise algorithm implemented, the output space of this approach may consist of either relevance degrees for all documents associated with the given query, or a ranked list (optimized permutation) of the documents. In the former case, the trained model directly optimizes the measure used to evaluate the performance of the solution. The difficutly in this stems from the fact that evaluation metrics, such as MAP and NDCG, are position-based, and therefore discontinuous and non-differentiable. Examples of algorithms directly optimizing evaluation metrics include SoftRank, which introduces randomness to the ranking scores of documents and calculates the expected value of NDCG@m as the objective function for learning to rank, and AdaRank, which utilizes boosting to optimize evaluation metrics. In the latter case, the loss function calculates the inconsistency between the algorithm's output permutation and the ground truth ranking. Different distributions on permutations can be used for ranking loss in this case. ListNet, for instance, makes use of the Luce model, one of the various proposed models for representing permutation probability distributions. Compared to previous algorithms, the listwise approach's overarching advantage stems from the loss function's capacity to solely consider the positions of all documents in the ranked list associated with the same given query.

3. EXPERIMENTS

3.1 Methods Studied

3.1.1 Discounted Cumulative Gain (DCG)

The aim of the methods studied is to find a number of ways of gathering quality for the entire ranking. An existing method that is popularly used in web search and other recommendation systems include the DCG, a measure of performance and non-binary relevance based on graded relevance of the given entity. Discounted gain is accumulated through the ranks taking a top-down approach âĂŞ the gain may be reduced, or discounted, at lower ranks. [X - standford]:

As further metrics such as average precisions are designed for binary metrics, DCG is designed to provide metrics for non-binary cases, where the utility is penalised by the rank:

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

* DCGk - The maximum number of entities that can be recommended

* rel - utility of a given document

The idea behind the DCG is to assume:

1. Documents of high relevance are more useful than documents that are marginally relevant.

2. Documents assigned a lower rank are less relevant, thus less useful for the user based on its likelihood to be.

3.1.2 Normalised Discounted Cumulative Gain (NDCG)

By taking the DCG of the ranking that is received, divided by the best possible ranking - we get normalized version of DCG. The graded relevance of a given entity is ranked between 0.0 to 1.0. [X - kaggle]. Where 1.0 represents the ideal ranking of an entity:

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

* IDCGk - the maximum possible (ideal) DCG for a given set of queries, documents and relevances

An ideal ranking works by first returning the documents with the highest relevance level, followed by the next, etc. The output computes the precision at rank for each retrieved relevant document. NDCG is computed by dividing the DCG by the Ideal Discounted Cumulative Gain (IDCG). The higher the nDCG, the better ranked list.

3.1.3 Precision & Recall

By observing the computed precision and recall at each rank, we are able to determine the search engine behaviour at a per query-doc level. Precision looks at a certain threshold in a ranking, we use further metrics such as MAP for an overall understanding of our ranking performance.

The F1 score is used through implementation for observation and measurement of accuracy using statistics precision p and recall r . Precision is defined as the ratio of true positives of all predicted positives (tp + fp). Recall (tp + fn) is defined as the ratio of true positives to actual positives:

$$F1 = 2\frac{pr}{p+r} \quad \text{where} \quad p = \frac{tp}{tp+fp}, \quad r = \frac{tp}{tp+fn}$$

F1 weights recall and precision the same. We use this measure to exercise the ranking system by observing its output as a good ranking system will aim in maximising both precision and recall.

3.1.4 Mean Average Precision (MAP)

During the tuning process of an algorithm, we use a common MAP metric to calculate the average of precision values at the ranks where relevant documents occur. Specifically, an indicator of the average of precisions in a single rank. This in contrast to Precision and Recall, observes the entire ranking process of all relevant documents where heavier weights are assigned to the start of the rank:

$$ap@n = \sum_{k=1}^{n} P(k)/min(m,n)$$

* P(k) - precision at cut-off k in the item list, equals 0 if k-th item is not followed upon recommendation

* m - is the number of relevant nodes; n is the number of predicted nodes.

* If the denominator is zero, P(k)/min(m,n) equals zero.

3.2 Dataset Statistics

Microsoft released two large scale datasets for research on learning to rank. They contain feature vectors extracted from query-url pairs with relevance judgement labels defined by a retired labeling set from Microsoft Bing, consisting of values 4 (very relevant) to 0 (irrelevant). The datasets include MSLR MSLR-WEB10K (1.2G compressed) worth 10,000 queries from random sampling.

Microsoft Dataset (MSLR âĂŞ WEB10K) not only provides data to create machine learning information retrieval

	Train	Test	Validation	Total
Query-doc pair	723,412	241,521	235,259	1,200,192
Unique Queries	6,000	2,000	2,000	10,000
Rel Score 0	377,957	124,784	121,522	624,263
Rel Score 1	232,569	77,896	75,815	386,280
Rel Score 2	95,082	32,459	31,910	159,451
Rel Score 3	12,658	4,450	4,209	21,317
Rel Score 4	5,146	1,932	1,803	8,881

Table 1: Dataset statistic, Fold1

Mean	120.02
Median	110
Max	908
Min	1

Table 2: Documents per query, Fold1

models but saves time and effort by providing data in easy to consume and process format taking care of five vital aspects:

· document corpus (with relevant query id and query document pairs)

· sampled documents to cover all the instances and not just belonging to one cluster

· features were already extracted and given as codes to work with

· meta information was provided that needs to be considered in document relevancy prediction

· 5 fold validation data to check the model accuracy and performance

Within each data file, a row corresponds to a query-url pair. Ordered as relevancy label of the pair, query id followed by all features:

0 $qid:1$ 1 : 3 2 : 0 3 : 2 4 : 2 ...135 : 0 136 : 0

2 $qid:1$ 1 : 3 2 : 3 3 : 0 4 : 0 ...135 : 0 136 : 0

Dataset contains 136 features along with a query-id(qid) and a relevance score, so we train the model to learn the relevance score of a particular query given the combined features of query-document pair.

The dataset is divided into five parts with somewhat equal query lengths for five-fold cross validation. Some basic statistics form folder 1 of the dataset are presented in Table 1 and Table 2 above.

3.3 Metrics & Results

RankNet is pairwise learning to rank approach using Gradient Descent to learn and update the weights and model parameters. Parameters such as batch size and no. of iterations were tuned to get the optimal output, which was measured in terms of NDCG.

$batchsize = 50$, $n\_iter = 1000$, $n\_units1 = 512$, $n\_units2 = 128$, $tv\_ratio = 0.0.66$, $optimizerAlgorithm = "Adam"$

*n_units are the no. of nodes in hidden layer 1 and 2

*tv_ratio is the ratio of training and validation set

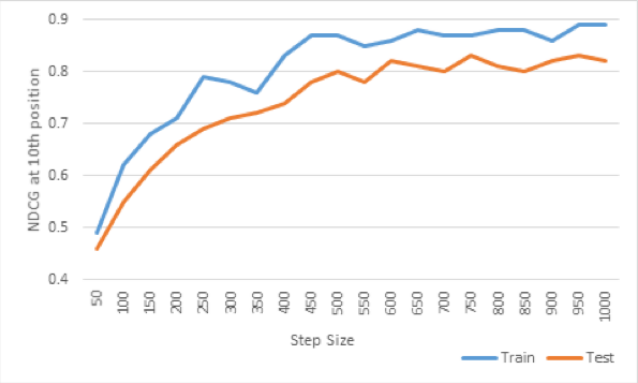


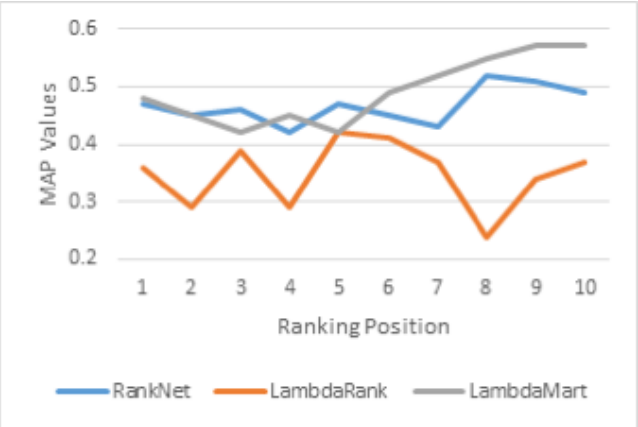
Figure 1: Gradient Boosting NDCG at 10th position

Gradient Boosting put together multiple weak learners into one ensemble model that focuses more on the data instances that tend to be misclassified before.

LambdaRank is derived from Ranknet but it calculates the gradient from candidate pairs, known as lambda and use that to swap the pair and promptly calculating NDCG. LambdaMART is based on LambdaRank but implements Gradient Boosting Trees where gradients are computed after each new tree to calculate the direction of loss function and ways to minimise that. It uses the family of MART models i.e. Multiple Additive Regression Trees.

3.4 Analysis of Results

3.4.1 RankNet, LambdaRank and LambdaMART



When more combinations are tested on LambdaMART, it improves the NDCG (Normalised Discounted Cumulative Gain) but it starts showing overfitting on test set. Increase in no. of trees also deviates the accuracy from validation set accuracy.

LambdaMART needs tuning of two parameters namely maximum number of leaves and learning rate, so different combinations were tried to get the optimal output which is calculated at NDCG. Grid search was used to try different combinations on a smaller set of data and the final model was put on use with the full validation and test set to out the values in table above.

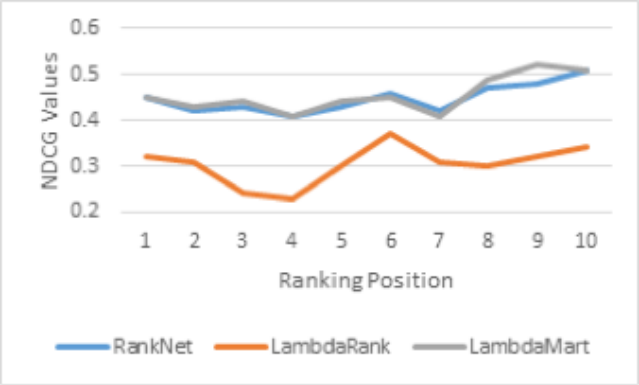


Figure 2: RN, LR and LM NDCG and MAP values

	Learning Rate				
Leaf	0.1	0.13	0.15	0.18	0.2
8	0.552	0.5529	0.5542	0.5541	0.5547
16	0.5561	0.5555	0.5558	0.5559	0.5548
32	0.5571	0.5574	0.5572	0.5568	0.5573
64	0.5583	0.5581	0.5581	0.5574	0.5563

Figure 3: Parameter tuning

3.4.2 *Classification Feature Engineering*

Prior to the feature engineering stage, the dataset is split into a training, validation and testing set. Cleansing of the data included splitting the feature list into multiple columns for separate interpretation and converting dataframe values to the correct data type to fit into the model once they were label encoded. Throughout the iterative process of analysing features based on relevancy label, features not useful were dropped to improve the overall ranking.

A total number of 50 features extracted from the vast amount of features offered in the MLSR dataset include:

> converted query term number, inverse document frequency, sum of term frequency, sum of stream length normalised term frequency, all tf*idf metrics, boolean and vector space models, language models for IR, page ranks and quality scores.

Our overall importance through feature extraction process is to sieve through the relevant features that impact the level of relevancy. Through this we can drop other non-correlated, non-relatable features that may add to the noise of accuracy in predicting unseen query-doc relevancy. To reduce the number of features, important streams were extracted for less important features and all streams extracted for more important features for ranking such as tf*idf calculations.

Using seaborn box-plots similar to those depicted in Figures 4 and 5 helped to identify the features that show trends with the relevancy score. An indicator of feature value performance and its impact on the level of relevancy.

3.4.3 *Classification Ranking*

Logistic Regression Classification model is used for predicting query-doc relevancy. Based on the features mentioned in 3.4.2, the query-doc is grouped in a Pandas data structure and ranked based on group qid and relevancy score. In the case a document with the same query-id has the same relevancy score as another, the highest prediction output from the relevancy categorical variables is selected by finding the max() of predict_proba probability output labelled as highest prediction, being the final decider of descending ranks on query id's with same relevance score.

This regression problem is known as the pointwise approach described in Section 2, where we gather the predicted score for highest classifier of the feature vector.

Figure 6 highlights the output for our final ranking algorithm. A number tests were undergone on 100,000 records versus the entire training and validation dataset. Our findings were such that the more samples that were used, the more feature input and an increase in lower rank prediction where fewer query-docs were being predicted a higher relevance score as the observation with smaller datasets was an increase in higher relevancy score. We were able to understand this behaviour by following through 3.1, putting our methods into action with our constructed ranking system.

3.4.4 *Classification Results*

DCG	3.5855583988200541
NDCG	0.54603583399465572
MAP	0.16666666666666669

Table 3: Fold 1 DCG, NDCG, MAP results (1,000 records)

We use DCG in order to emphasise documents of high relevancy appearing early in the result list - using logarithmic scale for reduction. The overall DCG performance for predicted relevance is 3.58 as a sum of all logarithmic ranking reductions. The DCG of this ideal ordering (IDCG) is 6.56652581313 computed within NDCG at 0.546. The NDCG results computed is difficult to use as NDCG only takes in partial relevance feedback therefore not a true ideal ordering of results is computed.

In order to validate the metric from multi-class classification and assess its performance through the Logistic Regression model, we compute the ROC area under the curve and plot the results in Figure 7. From analytical observation it is clear the level of true positives are increased for class 0 relevancy ranks followed by class 1 and a poor AUC score for relevancy classes 3 and 4.

From Table 4 it can be observed the model does an improved job at predicting records that are less relevant but struggles to maintain a decent F1 score for predicting those highly relevant query-docs. In recall, the occurrence of 0 for

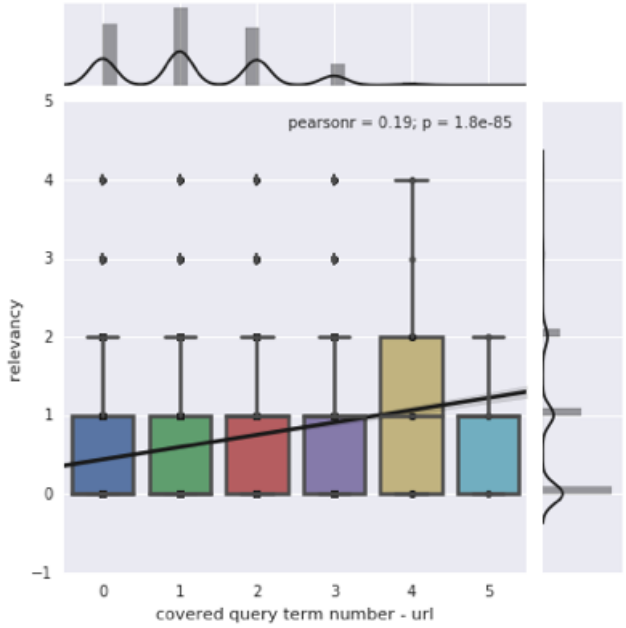


Figure 4: Covered query term number vs relevancy (1,000 records)

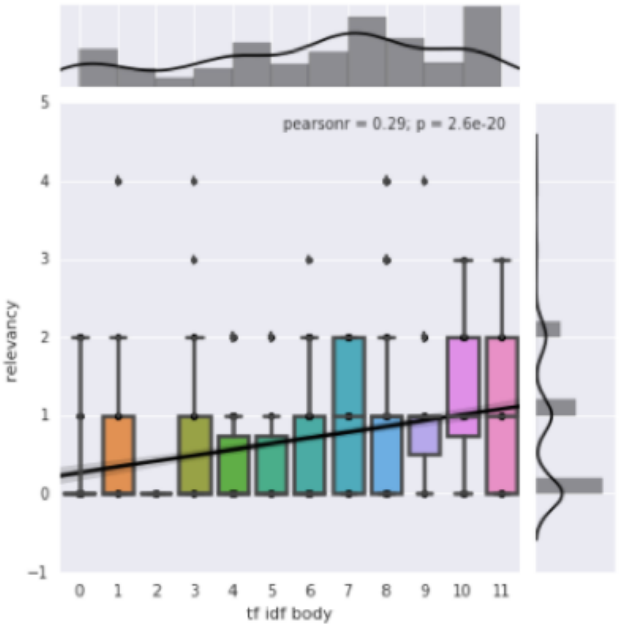


Figure 5: TF IDF vs relevancy (1,000 records)

index	relevance	qid	highest prediction
68	2	10	0.293
53	1	10	0.341
58	1	10	0.338
32	1	10	0.325
46	1	10	0.325
4	1	10	0.314
73	1	10	0.302
19	1	10	0.299
74	1	10	0.298
41	1	10	0.292

Figure 6: Classification Top 10 Ranking Test Output

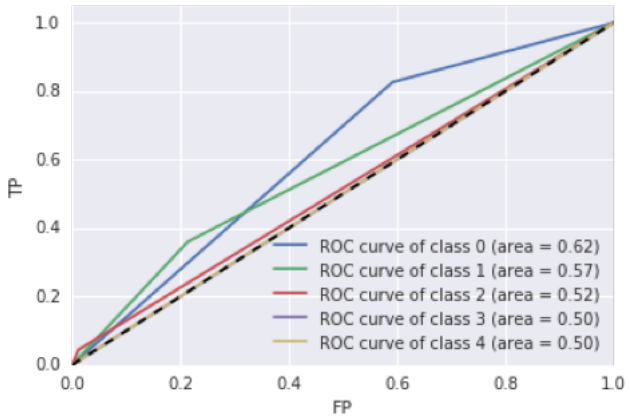


Figure 7: Classification Top 10 Ranking Test Output

	precision	recall	f1-score	support
0	0.56	0.89	0.69	121522
1	0.41	0.22	0.29	75815
2	0.38	0.01	0.02	31910
3	0.00	0.00	0.00	4209
4	0.12	0.00	0.00	1803
avg / total	0.47	0.53	0.45	235259

Table 4: Fold 1 Precision, recall and F1 score on validation set of LR Classifier

(tp + fn) certifies there are no positive cases of recall in the input data. For precision, there seems to be an improved precision in predictions but for category rank 3 indicates 0 for precision (tp + fp) where instances are predicted as negative. This correlates strongly with the feature modelling in 3.4.2 where a number of these features matched for relevancy rank ranges 0 - 3 and very few or non existent for relevancy rank 4. Overall, a better understanding of the difficulties that may arise when predicting multiple-class labels.

4. DISCUSSION & LIMITATIONS

All the algorithms reduce the ranking problem into further subsets. Pointwise algorithms like to reduce it to a regression or classification problem whereas pairwise approach reduce it to a pairwise classification. Benefits of these approaches are the ease of use and availability of existing tools to implement these algorithms. Both these approaches don't consider the relevant information ranking features and the loss function doesn't take into account the query as well as position information. Whereas listwise algorithms take into account the position information and train the model based on that thus creating ranking based on the query and the relevancy of document for that query. It gives better results but at the same time, listwise are hard and complex to implement.

For the top results for any search query, listwise algorithm performs the best but even pairwise tend to do better than the pointwise algorithms. Learning to rank problems are not and should be considered or reduced to normal ML problems as the results are query dependent and position specific so a general loss function will not be sufficient to dictate and judge the accuracy of the results. Data available for training and validating the models for Learning to rank algorithm lacks some important features that might be affecting the rankings on search engines. Click count is provided along with dwell time, but bounce rate and user journey can be some other relevant features as well. Heuristic plays an important role in Search Engine clicks through rate as the snippet of information called meta description, displayed with the search results, is often a deciding factor in someone clicking on the link or not. For this feature engineering was carried out on the data to take the meaning features and leaving out all the ones which will not impact the rankings that much. Some features can be combined together. In all a lot of work still needs to be done on feature engineering for the learning to rank models.

Ranking cannot and should not be simplified to be based on a loss function as the output space in listwise approach is a permutation of the relevant documents, which need further research to define the rankings based on the relevance score. Single ranking factor can't generalise for the all of query terms as the intention can differ for each query : navigational, transactional or just for information. And ranking models have to be different for all such queries as well. People searching for brand will be interested in their homepage but a relevant filter term with query should take them to that relevant section of the site rather than the homepage which has got the better page rank and other scoring metric as compared to that category page.

A limitation found during feature extraction is many of the features fed into the model linked to relevancies in the 0 - 3 range, where this was reflected in the classification predictions and the precision, recall and F1 score - not to mention the overall rank output. Had we undergone further feature extraction analysis and methods such as combined models, this could be improved upon.

Another limitation being the scale of available datasets is way too smaller than actual no. of actual searches performed every day, with biggest data set available containing just 30K queries whereas there are more than Billion online searches every day. Sampling is a hard process when available data is such a small proportion of the actual data that is available there.

136 features available in the Microsoft data set is a big list but it's far short of the optimal no. which is required to predict the actual ranking.

5. CONCLUSION

Various ranking models can be combined together to explore the opportunity of increasing the ranking prediction. Ranknet and LambdaRank do a good job as gradient boosting regressors but LambdaMART can outperform with its capabilities. Scores of one model can be fed into a second model to achieve even better results, though those are still not guaranteed to be optimal.

For our implemented ranking solution, further work could be carried out to improve the overall performance. Perhaps the use of a Gradient Boosting Classifier at this point would have sufficiently improved the recall, but would need to be excessively trained to find its optimum performance parameter. With the alternative of adding combined features in the feature vector space.

5.1 GitHub repo

The team's code can be found at https://github.com/thsol/irdm

6. ACKNOWLEDGMENTS

A big thanks to Dr. Jun and Dr. Emine for their efforts in teaching this module.

7. REFERENCES

[1] Christopher J. C. Burges RankNet: A ranking retrospective *Microsoft Research Blog, July 2015*

[2] Donmez, Pinar, Krysta M. Svore, and Christopher JC Burges. "On the local optimality of LambdaRank." *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval. ACM, 2009..*

[3] Burges, Chris, et al. "Learning to rank using gradient descent." *Proceedings of the 22nd international conference on Machine learning. ACM, 2005..*

[4] Xia, Fen, et al. "Listwise approach to learning to rank: theory and algorithm." *Proceedings of the 25th*

international conference on Machine learning. ACM, 2008. .

[5] Burges, Christopher JC. "From ranknet to lambdarank to lambdamart: An overview." *Learning 11.23-581 (2010): 81.*

[6] Li, Ping, et al. "McRank: Learning to Rank Using Multiple Classification and Gradient Boosting." *NIPS. Vol. 7. 2007.*

[7] Liu, Tie-Yan. "Learning to rank for information retrieval." *Foundations and TrendsÂő in Information Retrieval 3.3 (2009): 225-331.*

[8] "Mean Average Precision." Kaggle. N.p., n.d. Web. 17 Apr. 2017.

[9] "Normalized Discounted Cumulative Gain." Kaggle. N.p., n.d. Web. 17 Apr. 2017.

8. SOFTWARE

https://www.mathjax.org/

https://sourceforge.net/p/lemur/wiki/RankLib/ (RankNet, LambdaRank LambdaMart)