Learning to Rank - Feature engineering using a click model

Bachelor's Project Thesis

Klemen Voncina, s2900874, k.voncina@student.rug.nl
Supervisor: Dr. M.A. Wiering

Abstract: Effective ranking in information retrieval is done, in part, by proper feature engineering. This paper explores a comparison between the functions of a click model and a ranking function in information retrieval. It then uses the output of a basic bi-class click model as a feature for training a ranking model. Training both of these different approaches on data from a commercial search engine we find that click model performance improves as the threshold for what is a click becomes more stringent and that using the output of a click model as a feature for ranking performs empirically worse than without this added feature.

Keywords— Machine learning, Learning to rank, Click model, Web search, Information retrieval, Feature engineering

1 Introduction

In information retrieval, the primary objective is to present the user with the most relevant result based on their query. A good metric of performance for search engines would be to ask the question; "has the user's information need been satisfied?". A click model is something that attempts to answer this question using user browsing data connected to search queries. Research into click models follows one of several paths, either simulating user interactions or alternatively for use in ranking to attempt to guess, based on user interaction and other relevance scores, whether the page or document in question was relevant to the user or not.

In a click model (Chuklin, Markov, and de Rijke, 2015), several things are taken into account, depending on the complexity of the model and what we desire to predict using it. Generally the main considered set of events consists of four things; a user examining an object returned by a search engine, a user being attracted to an object's representation, a user clicking on an object in a search engine and the last being that the user's information need is satisfied (Chuklin et al., 2015). For the purpose of this model and given the datasets available, it is not possible to model whether a user has examined an object returned or if a user was attracted to it only if a user has interacted with it. So we consider the other two events; object clicks and attempting to determine whether the user's information need was satisfied based on that alone. The type of click model being referred to is called a click through model (Chuklin et al., 2015; Joachims, 2002)

Learning to rank (Pan, Luo, Tang, and Huang, 2011) is a subset of both the information retrieval and machine learning fields of study. A ranking function uses various relevance features computed from the document to attempt to predict that document's relevance to the user, then score and order it appropriately. This means that the performance of a ranking algorithm may be improved through either better error or performance metrics or by attempting to improve the features through feature engineering.

In light of this, the research question guiding this thesis is the following; How does a ranking algorithm compare to a click model and is it sensible, in terms of ranking performance, to use a click model's output as a feature in ranking?

Features used for solving the learning to rank problem include such relevance scores as TF-IDF and Okapi-BM25 (Robertson and Zaragoza, 2009) which are both measures of relevance derived from term frequency. This is to say, for TF-IDF for example, the score is calculated by taking the number of times a query term appears in a document and dividing it by the total amount of times it appears in all the documents the query returned. These are then weighed against human judged relevance labels for a given query-document pair.

Both of these modelling types can be trained on similar data as in both cases, the end goal is to determine what feature or group of features holds the greatest sway in predicting whether a document will be relevant to a user and if viewing that document will sate their information need.

The intended contribution of this paper is to present a comparison between a click model and a ranking function, then evaluate whether it is sensible, in terms of ranking performance, to use the output of a click model as a feature in ranking. The proposed process requires the training of two models, a ranking model and a click model, then testing them. First, testing each separately then using the output of the latter as a feature for the former. The remainder of this paper will be structured in the following way. First an in depth analysis of the purpose of a click model and a ranking algorithm will be made along with a more detailed explanation of the contents of the LETOR dataset. Following that, the methodology behind each of the two classifiers will be presented as well as the exact setup. This will be proceeded by the experimental results and then a conclusion drawn from the experimental results obtained.

2 Background

2.1 Learning to Rank Data

Learning to rank data sets are formatted in the following way, each data point is a query-document pair positioned in N dimensional space where N is the number of features. Each of the data points has the following details; a relevance score, a query ID followed by a feature vector of N features as shown in Table 2.1.

An important concept to grasp when considering a learning to rank data-set is the concept of a query-document pair. A query document pair is one data point in a learning to rank data-set. A document does not have a relevance score compared to all the other documents at all times, it has a relevance score relative to a query. As an example consider two documents m and n relative to queries A and B; for both queries both documents are returned, document m may be very relevant to query A while document n is not relevant at all, conversely document n may be very relevant to query B while document m is only slightly relevant. This is indicative of the fact that the goal is not to find the most relevant document for everything in the entire collection of documents, but we wish to use the ranking model to bring to the forefront the most relevant document for a particular query instead.

Rel. Lab.	Query ID	Feature Vector
1	quid:1	1:0.03 2:0.66 ...N:M
0	quid:1	1:0.03 2:0.00 ...N:M
0	quid:1	1:0.00 2:0.16 ...N:M
2	quid:2	1:0.99 2:0.00 ...N:M
0	quid:2	1:0.05 2:0.75 ...N:M

Table 2.1: The format of a learning to rank training set

In addition to document relevance scores being assigned to the document relative to the query, most of the features found in the query-document feature vector are computed about the document relative to the query they are returned by as well.

A learning to rank training dataset consists of a set of labels, query IDs and feature vectors for each data point. These feature vectors can vary widely in size (Qin and Liu, 2013; Qin, Liu, Xu, and Li, 2010) and generally consist of many of the same features being computed for several separate fields of a document. Most of these 'relevance features' as they are referred to are metrics derived from the concept of term frequency, or how many times a term or phrase from the query appears in the document as compared to to total number of words in that document. The most basic of these is a measure called TF-IDF (term frequency, inverse document frequency) which measures how many times a term appears in a document versus how many times that same term appears in all the documents retrieved by the query.

As previously alluded to, there are some features that are computed for a certain document based on the query that returns it such as the example with the term frequency metrics. There are, however several features about a document that can be independent of the query. Some examples of features that are query independent and are static per document from the LETOR 4.0 dataset are the number of outlinks or the number of slashes in the

URL. Features like this are not dependent on query terms at all and can therefore be considered query independent features.

It is important to note that these kinds of features exist, because dwell time of a document is generally a document specific feature rather than a query-document computed relevance feature. It is likely that, given an extensive amount of metadata on user interaction per query and subsequently per document it would be possible to have query-document specific user dwell times, however this data is generally not available in this format.

2.2 Ranking Function

Ranking algorithms or functions fall into several different categories based on what is considered a training step. There are point-wise ranking algorithms, pair-wise ranking algorithms and list-wise ranking algorithms (Ibrahim and Landa-Silva, 2017). Point-wise ranking algorithms operate on one query document pair at once as a learning instance, meaning that a query document pair is given a relevance rating all of its own independent of any other documents returned by the query. The resulting ranking is then a series of independently weighted documents for a certain query. A pairwise ranking algorithm has two steps, first it performs something called a par-wise transform, taking two query document pairs, comparing their labels and, if they have different labels they can be considered a 'pair'. The reason the labels have to be different within a pair is because one has to be considered better than the other, otherwise re-ordering items inside that pair or query document pairs is entirely without purpose. A pair is then used as a learning instance in the algorithm. Lastly, list-wise ranking considers all the query document pairs in a given query as a learning instance and every value judgment that the ranking algorithm makes takes into consideration all the other items in the list.

As would be expected, therefore, point-wise ranking is the fastest in implementation but also the least accurate as a result of not comparing any query document pairs to each other, list-wise ranking is the slowest in implementation as every action considers every query document pair in the list and lastly; pair-wise ranking is a middle of the road technique that does some comparison between query document pairs for a single query but as it only compares two documents to each other at once, its speed is also somewhere in the middle of the other two.

Given that the purpose of a ranking algorithm is to find weights for features such that the documents are ordered in or as close to the optimal order, we would like to consider how this is accomplished. A ranking model with a linear ranking function output, for example would find a vector in k dimensional space such that, when all the points in the training set are mapped onto this vector using perpendicular lines, the direction of the vector matches the order of the items in the training set. For explanatory purposes, we will call the candidates and non-candidates for this vector w_1 and w_2. As shown by Figure 2.1 and Figure 2.2, w_1 is not a good candidate for a ranking function, whereas w_2 most definitely is as it matches the order of the items.

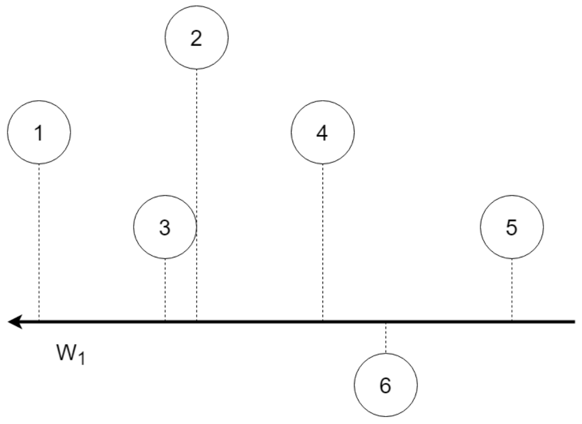

Figure 2.1: Showing a bad or non-candidate for ranking function w, w_1. Perceived order from ranking function w_1 is $1, 3, 2, 4, 6, 5$.

The product of a ranking function or a learning to rank algorithm is a ranking model which is used in web search as shown in Figure 2.3. In this case, we are specifically interested in the query-document pairs that are used as input for the ranking algorithm and what features are contained therein as well as the process of query-document feature computation based on returned documents for a user query.

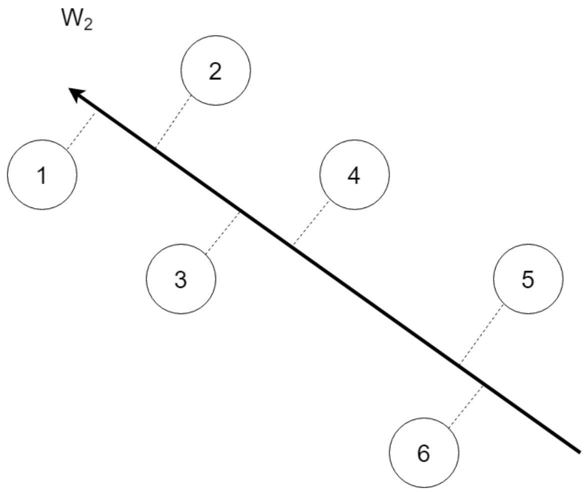

Figure 2.2: Showing a good candidate for ranking function w, w_2. Perceived order from ranking function w_2 is $1, 2, 3, 4, 5, 6$.

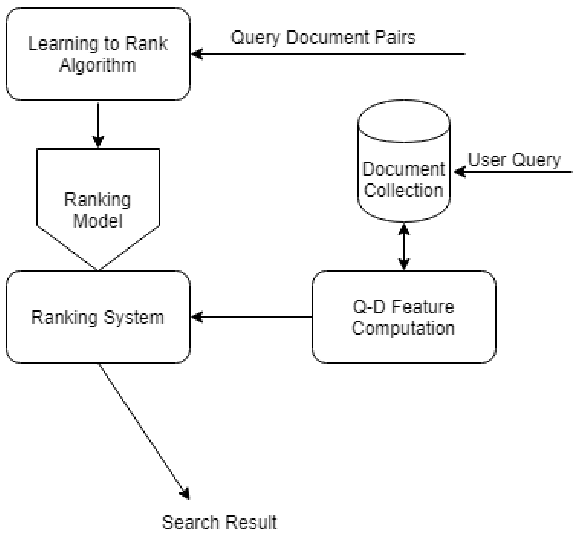

Figure 2.3: How learning to rank is integrated into web search

2.3 Click Model

A click model, depending on its exact intended purpose can be used for the following things; predicting whether a user will interact with a document, simulating user interaction in general and gauging whether the user has achieved their desired goal. This requires a large amount of user browsing metadata, or implicit data about documents. Several click model types are proposed which cater to the various goals of click modelling such as user browsing models or click chain models (Chuklin et al., 2015) . This paper considers only one type of model, the click through rate model, specifically a subsection of this being the document based click through rate model. This means that the model that is built attempts to predict whether a document will be clicked on by a user given the query-document pair relevance features.

Click models attempt to predict the probability of something being clicked. This can be replicated by using a bi-class classifier that distinguishes between 'likely to click' and 'unlikely to click'.

3 Method

This paper proposes that the output of a click model trained on query-document data and the dwell time of users on that document may be a viable feature to use in training a ranking function on the same document set. The reasoning for this is that user dwell time on a document can be taken as an implicit rating of a document's relevance to a search (Chuklin et al., 2015). This quality distilled in a feature may provide something similar to the assigned relevance rating that is provided in the learning to rank dataset by human judgment.

3.1 Ranking Algorithm

The proposed method involves training a ranking model using a ranking algorithm called RankNet (Burges, Shaked, Renshaw, Lazier, Deeds, Hamilton, and Hullender, 2005). RankNet is a pairwise ranking method that uses a neural network as its backbone (Pedregosa, Varoquaux, Gramfort, Michel, Thirion, Grisel, Blondel, Prettenhofer, Weiss, Dubourg, Vanderplas, Passos, Cournapeau, Brucher, Perrot, and Duchesnay, 2011; Buitinck, Louppe, Blondel, Pedregosa, Mueller, Grisel, Niculae, Prettenhofer, Gramfort, Grobler, Layton, VanderPlas, Joly, Holt, and Varoquaux, 2013; Schmidhuber, 2015). The features that differentiate this from a 'normal' multilayer perceptron are twofold, the pairwise transform (Ailon, 2012) of the training data and the purpose of the .

Algorithm 3.1 describes the pairwise transform operation that is performed on the data before it is passed to the ranking algorithm.

Algorithm 3.1 Pairwise Transform Function

```
Input: A $k$ length list of labels $y$ and a $k$ length list of feature vectors $X$ with the same query ID
Output: A list of pairwise comparisons at most $\frac{k^2+k}{2}$ items long
  $yPairs \leftarrow emptyList$
  $xPairs \leftarrow emptyList$
  for $i$ to $k$ do
    for $j$ to $k$ do
      if $y_i \neq y_j$ then
        $yPairs$ append $y_i - y_j$
        $xPairs$ append $X_i - X_j$
      end if
    end for
  end for
  return $(yPairs, xPairs)$
```

This pairwise transform, given k items returns at most $(k^2 + k)/2$ items, however given that the dataset only consists of three classes this maximum number of returned items will never occur when $k > 3$. Because the pairwise transform is applied on query groupings before training, it is then no longer necessary to present the ranking algorithm with batches of training data corresponding to the number of query-document pairs for each query, instead the batch size may now be adjusted to arbitrary size.

Training in RankNet is then done on these pairwise transformed learning instances. This means that the RankNet algorithm becomes a comparator discerning between 'better' and 'worse' in a pair of items. The fact that items with equal labels are not subject to the pairwise transform works beneficially here as it reduces the number of categories to two rather than three. In addition to this having a 'same relevance' category would be detrimental to ranking as a stalemate between two items is not desirable.

3.1.1 Evaluation Metrics - NDCG

Discounted cumulative gain (Wang, Wang, Li, He, Chen, and Liu, 2013) is the measure of improvement of a document ranking compared to the ideal ranking. Each position in a set comes with its own 'multiplier' given by a logarithmic function meaning that items lower on the list will be worth much less. This has the added benefit of hugely rewarding moving a very relevant document from lower in the list to higher in the list.

Normalized discounted cumulative gain is calculated in the following manner:

$$NDCG@P = \frac{DCG@P}{IDCG@P} \tag{3.1}$$

$$DCG@P = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{3.2}$$

$$IDCG@P = \sum_{i=1}^{|REL|_p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{3.3}$$

In the above formula; p is the number of items being considered, rel_i is the relevance score of the document at position i and $|REL|_p$ refers to the fact that, for this computation (Equation 3.3), the ideal ranking of the top p documents in the set is used.

If the top P items in a list are ordered in the optimal manner when the metric is being applied, then the following is true; $DCG@P = IDCG@P$. In the event that the relevance scores are only either zero or one, and the top P documents are all relevant, then the same is true.

Normalized discounted cumulative gain is used instead of discounted cumulative gain in learning to rank because it is expected that queries will not always return the same number of documents as each other. This means that to be able to compare the score between two queries, the score must be normalized against the ideal discounted cumulative gain across a certain number of elements. NDCG normalized over P items would be noted as $NDCG@P$.

This NDCG score is used as the loss function we wish to minimize in stochastic gradient descent when adjusting the weights of the network (Kingma and Ba, 2014).

3.2 Click Model

In this section a click model is proposed. A very simple multilayer perceptron that distinguishes between two classes; 'user likely to click' or 'user not likely to click'. Similarly to how the RankNet algorithm works, this uses a neural net with two layers of hidden rectified linear activation units is used. In the same manner as before; adam (Kingma and Ba, 2014) a method of stochastic gradient descent is used as the optimization algorithm.

In order to up the stakes, three different click models will be created, all using the same training data, however the dwell time will determine the labels differently in each of the three scenarios. The three click models that are proposed use the following three 'thresholds' or cutoff points for determining what is considered a click and what is not considered a click. Keep in mind that the goal of the click model is not only to determine whether the user will click on a document but also whether the user subsequently found that document useful. As we know from previous literature, a user will likely examine a document that is highly ranked but not relevant to the search, however, given that they generally do not find what they are looking for, the user will leave shortly after.

The data extracted is compounded average dwell time per user on a document. As these are averages, and a user is unlikely to interact with a document very long if they do not find what they are looking for, it stands to reason that the longer a user spends on a document, the more relevant they found it.

Given this information, the three different proposed click models use three different cutoff points to determine what is a click. First, any dwell time above zero is considered a click. This postulates that any interaction with a document signifies that a user is interested in the information contained therein and the interaction means they have found what they were looking for. The second proposed click model uses 10 seconds as a cutoff point. This model postulates that any user interaction longer than 10 seconds is considered as an indicator that a document has been relevant to a user. Lastly, the third click model uses 25 seconds as a cutoff point, postulating that any document with average user interaction lasting 25 seconds or longer is considered a document that has fulfilled the user's information need.

These relevance cutoffs are what replaces the relevance label on the same base dataset that is used for ranking. To be clear, the original relevance label of the dataset as used in the ranking model and as presented in Table 2.1 is entirely removed and not considered for this portion of training. It is replaced with the binary consideration described here. For example, regardless of the relevance label that was present on the document beforehand, now the labels are binary and determined by dwell time. The label is either a 0 or a 1, 0 if something is not considered a click, and 1 if something is considered a click. As an example, consider the following; in the 10 second cutoff click model a query-document pair A with a previously assigned label of 0 has an average user dwell time of 15.2 seconds, its label therefore is 1. Conversely, document-query pair B with a previous relevance label of 2 which has an average user interaction time of 5 seconds, now has a label of 0. Note that this is an example and not an actual outtake of the dataset.

Contrasting again with the previous ranking model, here the data is not grouped by query when being considered for training. This is done for one key reason. Whereas before, the labels assigned to documents were query dependent, meaning that the label applied to the document only when it is being considered in tandem with a specific query, the dwell time is document specific and entirely independent of query. This means that the training can be done on arbitrary selections of data, rather than using the queries for grouping.

3.3 Click Model as a Feature

For each of the three click models created, the click model is used to predict whether or not a user will interact with a document based on the threshold. The output of this is then added as the 47th feature in the LETOR 4.0 dataset. This is then considered the transformed data with the additional feature which is later used in ranking again.

In application, if this method is proven effective, this would affect real search engine integration in the following way; as compared to Figure 2.3, we would be adding one additional element between the document collection and the ranking system, namely the click model. It would, in fact, sit between the Q-D Feature computation node and the ranking system as the click model is dependent on the current set of query-document features that are already being considered.

The hypothesis is therefore posed that using the output of a bi-class classifier trained on aggregate user browsing data of documents as a feature in training a ranking function will have an overall positive effect on the performance (measured in

NDCG@100) of the ranking model.

4 Experimental Results

4.1 Datasets

All the results in this section are obtained using two benchmark data sets from the LETOR 4.0 collection. Specifically, MQ2007 and MQ2008. The two data sets in question are summarized in more detail in Table 4.1. MQ2007 and MQ2008 have a combined total of 2476 unique queries and 84834 query-document pairs.

Dataset	Features	Queries	q-d pairs	Labels
MQ2007	46	784	15211	0,1,2
MQ2008	46	1692	69623	0,1,2

Table 4.1: Properties of LETOR 4.0 datasets

Both these datasets are, by default split into 5 subsets for cross validation. In this case, 5 fold validation is used for both datasets.

4.2 Click Models

4.2.1 Dataset Transformations

For the purposes of the following tests, the MQ2007 and MQ2007 datasets were transformed in the following ways; the labels were replaced with value judgments of either 0 or 1 (0 for no click and 1 for click). These value judgments were assigned based on average user dwell time on a certain document and one of the three mentioned thresholds. First all user interaction was considered a valid or relevant click, then anything over 10 seconds average user dwell time, then anything over 25 seconds average user dwell time.

4.2.2 Experimental Data

Click Threshold	Mean	SD
All	0.54	0.00
10s	0.74	0.01
25s	0.94	0.00

Table 4.2: Click model accuracy statistical summary MQ2007

Click Threshold	Mean	SD
All	0.54	0.01
10s	0.81	0.02
25s	0.93	0.00

Table 4.3: Click model accuracy statistical summary MQ2008

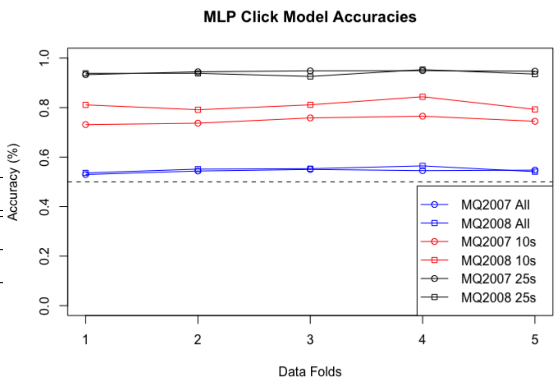

Figure 4.1: MLP click model results

The click models were trained using a MLP with two hidden layers with 128 and 64 rectified linear units in each layer respectively, while the adam optimizer algorithm used default parameters. These parameters were determined to be optimal after a coarse parameter search. The output of the MLP is a class judgment, either something is a predicted click or it is not. As is evident from the results, especially from Figure 4.1. The click model is a bi-class classifier meaning that depending on the data's bias to one side or another (whether there are more elements than half in either class), the worst performance that the classifier would achieve by simply guessing at random would be 50%. There is a mark on the plot to indicate this. Based on this, clearly the click model that counts every single user interaction as a click has no relation to the features of the dataset. As shown by both Figure 4.1 and Tables 4.2 and 4.3, the click model shows a marked improvement in accuracy when the requirements for something being considered a click are tightened and any user interaction shorter than 10 seconds is dismissed as irrelevant. Further stepping up the threshold for what is considered a click to 25 seconds of user interaction shows even clearer improvement, clearly there is a correlation between features of a document and some aspect of why a user would spend on average more than 25 seconds on a document.

While the low standard deviation of the click model with no threshold hints towards a 'lucky guess' scenario where there may be some correlation between any click interaction at all and a document's features. The classifier's performance is so poor, however, that this fact alone makes it useless by default. While the 25 second threshold model shows certain promise, the 10 second threshold model may still lead to confounding output down the line given its middling performance and score.

4.3 Ranking Models

4.3.1 Dataset Transformations

As is implied by the nomenclature, the baseline test leaves the dataset entirely unchanged for an initial test of ranking performance. The consequent tests, however, were performed using a modified version of the dataset. For each of the three proposed click models, the dataset was transformed once. All the labels stayed in their original orders, however the feature vectors of each data point were expanded by one to make room for the click model output, the ranking is now being performed with 47 features instead of 46.

4.3.2 Experimental Data

Dataset	Mean	SD
MQ2007	0.402	0.03
MQ2008	0.416	0.03

Table 4.4: Baseline NDCG@100 measures

First, we aim to establish a baseline performance for the ranking algorithm on the desired datasets so it can then be compared to the performance once the additional feature is added by means of the proposed click models. All ranking functions described here were trained using an implementation of the RankNet algorithm using two hidden layers of rectified linear units numbering 512 and 128 per hidden layer respectively. Again, these parameters were determined by a coarse parameter search and adam, the optimizer, is again using default parameters for this.

Dataset	Mean	SD
MQ2007	0.339	0.02
MQ2008	0.366	0.03

Table 4.5: NDCG@100 for ranking using All click model

Dataset	Mean	SD
MQ2007	0.3353	0.030
MQ2008	0.4001	0.028

Table 4.6: NDCG@100 for ranking using 10s threshold click model

Dataset	Mean	SD
MQ2007	0.3209	0.032
MQ2008	0.3849	0.030

Table 4.7: NDCG@100 for ranking using 25s threshold click model

What has become clear from the data acquired is that the click model does seem to have an impact on ranking performance, however it is immediately apparent that this influence is not positive. This can be confirmed by performing a paired t-test on the mean NDCG@100 scores obtained. The results of these tests conclude there is a statistically significant difference in means. These statistical tests can be reproduced given the data in Tables 4.4 through 4.7

There are some anomalies in the first fold of cross validation for two of the 10 second and 25 second click models trained on MQ2008 data, however this anomaly seems to entirely disappear by the third fold of testing. This likely means that the small size of the dataset allowed for some over-fitting early on, however this appears to have evened out towards the end. The overall mean of the data obtained from those two tests is still smaller than the baseline tests in a statistically significant manner. The larger MQ2007 dataset seems to perform overall consistently worse with the click model output than the considerably smaller MQ2008 dataset.

What further shows that the click model does have some impact on the ranking is that despite

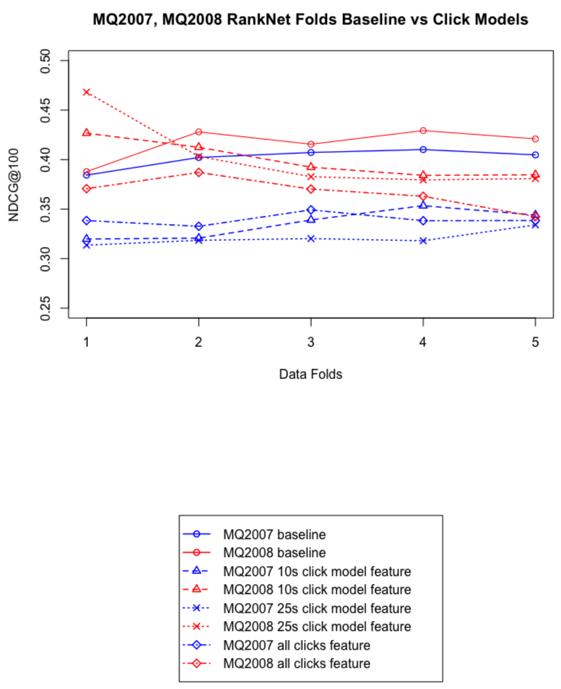

Figure 4.2: Comparison of all the ranking models with and without the additional features added by the click models

the five fold validation, the results are consistently worse. The statistically significant nature of the difference in means shows this overwhelmingly meaning that while overall the click model is a confounding variable it appears to succeed just often enough in the wrong ways to skew the end results.

5 Conclusions and Further Research

In conclusion, this paper shows that a ranking algorithm and a click model differ in the scope of their consideration. While a click model, at least the one explored here, covers the scope of a document. It does not achieve the granularity of query level representation. With aggregate interaction data, however, it can still perform some useful overview like finding the 'cream of the crop' of relevant documents quite reliably. This comes with the drawback that outside this application, the performance of the click model is mediocre at best. A ranking function, on the other hand must consider documents in the context of queries in order to achieve its best performance. The portion of the exploration delving into the possibility of integrating the two methods by using the output of a click model as a feature in query-document pair ranking has shown, with overwhelming evidence that using document level relevance judgments in a more granular, query-document oriented ranking environment serves as a confounding variable and, in fact, deteriorates the performance of the ranking algorithm.

Future explorations should be focused on one or both of the following; using a continuous output click model rather than a classifier and the possibility of obtaining a dataset where the URL/document dwell time is aggregated at a query dependent level and not simply on a document level of granularity.

References

Nir Ailon. An active learning algorithm for ranking from pairwise preferences with an almost optimal query complexity. *Journal of Machine learning Research*, 13:137–164, 2012.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gred Hullender. Learning to rank using gradient descent. *Proceedings of the 22nd International Conference on Machine learning*, 22, 2005.

Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. *Click Models for Web Search*. Morgan & Claypool, 2015. ISBN 9781627056489.

Osman Ali Sadek Ibrahim and Dario Landa-Silva. ES-rank: Evolution strategy learning to rank approach. *Proceedings of the Symposium on Applied Computing*, pages 944–950, 2017.

Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM. URL `http://doi.acm.org/10.1145/775047.775067`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Yan Pan, Hai-Xia Luo, Yong Tang, and Chang-Qin Huang. Learning to rank with document ranks and scores. *Knowledge-Based Systems*, 24 (4):478–483, 2011.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013. URL `http://arxiv.org/abs/1306.2597`.

Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval Journal*, 2010.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, April 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL `http://dx.doi.org/10.1561/1500000019`.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. A theoretical analysis of normalized discounted cumulative gain (NDCG) ranking measures. 2013.