Optimisation methods for ranking functions with multiple parameters

Michael Taylor
Microsoft Research
7 JJ Thomson Avenue
Cambridge CB3 0FB, UK
mitaylor@microsoft.com

Hugo Zaragoza
Yahoo! Research
Ocata 1
Barcelona 08003, Spain
hugoz@es.yahoo-inc.com

Nick Craswell
Microsoft Research
7 JJ Thomson Avenue
Cambridge CB3 0FB, UK
nickcr@microsoft.com

Stephen Robertson
Microsoft Research
7 JJ Thomson Avenue
Cambridge CB3 0FB, UK
ser@microsoft.com

Chris Burges
Microsoft Research
One Microsoft Way
Redmond, WA 98052, USA
cburges@microsoft.com

ABSTRACT

Optimising the parameters of ranking functions with respect to standard IR rank-dependent cost functions has eluded satisfactory analytical treatment. We build on recent advances in alternative differentiable pairwise cost functions, and show that these techniques can be successfully applied to tuning the parameters of an existing family of IR scoring functions (BM25), in the sense that we cannot do better using sensible search heuristics that directly optimize the rank-based cost function NDCG. We also demonstrate how the size of training set affects the number of parameters we can hope to tune this way.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Storage and Retrieval—*information search and retrieval*

General Terms

Experimentation

Keywords

evaluation, optimisation, effectiveness measures, ranking, scoring

1. INTRODUCTION

Traditional retrieval functions have very few *free* parameters, but nevertheless these parameters need to be set to some value, and this choice affects the resulting performance. In order to determine the optimal value of the free parameters, experiments are typically run testing very many parameter values on some training set and selecting those that lead to the best performance with respect to the measure of choice. Such an exhaustive search for the optimal parameter values is both simple and extremely effective: it can be applied to any combination of retrieval function and performance measure. However, the number of values that need testing grows exponentially with the number of parameters. For this reason, exhaustive searches can only be carried out for very few parameters. Beyond that, exhaustive search is impossible and we need to use heuristics or a learning algorithm to direct the search for optimal parameter values.

This results in a difficult situation well known to most IR researchers: we add features to our retrieval functions to improve their performance, but in doing so we increase the number of unknown parameters, which makes it harder to find good parameter values, which in turns hurts performance. In particular we consider that negative results (e.g. *adding this feature did not help...*) make sense only if we believe that we have reached the global maximum of the new ranking function with respect to the performance measure of choice.

For these reasons the lack of efficient learning algorithms for retrieval functions (and retrieval performance measures) seems to us one of the bottlenecks of IR research today. In this paper we do not propose a solution to this problem, but rather, we discuss and test two approaches available to us today: traditional greedy searches, and an extension of the gradient descent approach recently proposed in [4]. In this paper, we apply these to training sets of up to 2048 queries.

Training set size

What is a typical training set size? This depends very much on the environment of the experiment. Our evaluation data is taken from a commercial Web search engine; it is reasonable to suppose that all such search engines typically collect and use very large training sets, considerably larger than those used here. On the other hand, a common TREC sit-

CIKM'06, November 5–11, 2006, Arlington, Virginia, USA.

uation is to have 50 queries to train on, at the lower end of our range.

Our guess is that there are relatively few environments in which very large training sets are feasible – outside of the Web search engines, and possibly a few other public search services and the intranets of a handful of large companies. In many environments, the question of whether one can reasonably train on less than (say) 100 queries may be critical to any training exercise. The choice of training set sizes in this paper is informed by such considerations.

Overview of the paper

First we summarise some fundamental points about parameters and optimisation, including a brief review of some of the methods used in search engine tuning. Next we discuss the measure of effectiveness we use: a primary metric which we really wish to optimise, and a substitute metric used in the tuning process. Then we describe the tuning methods and procedures, for which we require gradients (derivatives) of the metric with respect to the parameters of the model. The BM25F ranking function, which is a component of the search engine, has non-linear parameters; the gradients for this component are established in the Appendix. In sections 3 and 4, we present experiments and results, with different size models (number of parameters) and different size traning sets.

The primary contributions of the paper are as follows:

- an empirical analysis of the interaction between size of model (number of free parameters) and training set size; and

- an investigation of the optimisation of the parameters of BM25F, including derivation of the gradients as indicated.

In the course of this work, we compare two alternative methods of optimisation, and show that a relatively fast machine learning method can perform just as well as a naïve search of parameter space. We also show that significant benefit can be gained from using a large number of weak features, over and above the usual range of features used in typical TREC web track work.

2. PARAMETER OPTIMISATION

A model, in this case a ranking model for information retrieval, will have some number of free parameters. If the model has too few free parameters, then its effectiveness is limited, because even given a large amount of extra training data it cannot improve. It has too few degrees of freedom. If the model has too many free parameters, the danger is overfitting. Overfitting gives very good performance on the training data, but the system does not generalise to new examples, so will perform poorly on a test set of queries. [1]

A typical machine learning approach would be to err on the side of having a powerful model with many parameters, to take advantage of the available training data, and to then employ additional techniques to avoid overfitting. One such technique is early stopping, to which subject we will return in section 2.5.

A fundamental question for information retrieval is whether these machine learning approaches, using large numbers of parameters and large amounts of training data, can give better retrieval effectiveness than approaches with few parameters. A significant barrier to study in this area has been finding efficient methods for exploring large parameter spaces, because rank-based effectiveness metrics do not lend themselves to efficient tuning. This is discussed in detail in the remainder of the paper.

There are two general approaches used by the IR community to handle such tuning. One is to try many parameter settings and evaluate the objective function (such as mean average precision) at each parameter setting: potentially a very expensive procedure for many parameters. These approaches can be further sub-divided into a) approaches that involve full exploration of a low-dimensional grid of parameters, for example a full 3-D grid [7], and b) heuristics that allow incomplete exploration of more dimensions [15] [2]. These latter methods are used as a baseline in this paper, and further details are presented in section 2.2.

The other approach is to choose a metric that does not depend on calculating an overall ranking [6, 5], but that can be more easily optimised, for example by gradient descent [4]. Such approaches are potentially much faster, but have the potential disadvantage of mismatch between the metric being optimised and the rank-based metric that better reflects user satisfaction [12]. This is one of the questions addressed in the present paper.

In the work discussed below, we use as our primary metric only the rank-based NDCG, defined below. However, we make no assumptions about the primary measure, and our methods could be used with any single-point or rank-based measure, such as for example Mean Average Precision.

Some related work

A number of other approaches to this problem have also been used. Methods based on SVMs, for example [10], typically have a time-complexity which depends on the size of the training set rather than the number of parameters. Such methods also interact with the choice of metric: for example Joachims [10] uses metrics relating to a singe point in the ranked output such as Precision at rank k. He extends his method to one rank-based measure, the area under the ROC curve – his argument depends on being able to define this area in terms of the probability of pairwise error.

Wanting to consider cases with a large amount of training data, we do not pursue the SVM option here.

Generalisability

Optimising the parameters of a model for a particular set of conditions raises questions of generalisability, beyond the overfitting question already raised. Either or both of the set of documents or of topics may be special in some way, or systematically different from the target real environment in which the system will be deployed. A similar problem is frequently observed in a TREC environment – some choice of methods or parameters which worked well on last year's TREC data will often fail on this year's task.

This is a general problem, not addressed in the present paper (we assume some sampling process in generating the training and test topics, and a fixed document collection). We expect, for purely statistical reasons, to overfit because our training set is too small, but not because of systematic differences. To some extent our assumptions mirror the situation from which our data is taken, a general web search engine. We might expect the overall character of the documents and topics to change over time, but probably gradually. However, we recognise the existence of the general problem.

2.1 IR Effectiveness measure: NDCG

The corpus used in this paper has 5 levels of relevance judgement: four relevant levels and one irrelevant. A suitable ranking effectiveness measure that makes use of the multi-level relevance judgements is NDCG (normalised discounted cumulative gain) [9]. The four positive relevance grades are assigned rather arbitrary gains of 31, 15, 7 and 3 respectively. The rank-based discount function is $\frac{1}{log(1+j)}$ for rank j; however, the calculation is truncated at rank 10 (in effect the discount function goes to zero after rank 10). As usual for NDCG, we calculate DCG for each query, by stepping down the ranks accumulating the discounted gain at each rank; we also calculate the best possible DCG for this query, derived from an ideal ranking in which a more-relevant document always precedes a less-relevant document. DCG is normalised per query by this maximum value, and the resulting per-query NDCG is averaged over queries.

Another feature of the data is the relative sparsity of relevance judgements: only shallow pools have been judged. This raises questions about how to evaluate ranking measures such as NDCG, given that there may be many unrated documents early in the ranking resulting from any changed ranking function. We might consider either treating unrated documents as non-relevant (this is the usual IR convention) or evaluating the measure only on rated documents. In the experiments presented here, we adopt the former usual convention when reporting NDCG on a test set.

2.2 Optimising NDCG

Most IR performance measures of interest, including NDCG, are defined in terms of the *rank* of the documents and not their score. Computing the rank involves a *sorting* operation on the scores, which is not continuous, and for which no suitable gradients have been proposed. For this reason, it is clearly difficult to use gradient descent optimization techniques.

Line search

Given this difficulty, it is natural to use techniques that do not depend on gradient, but instead only on the actual computation of NDCG. An obvious candidate is the general class of optimisiation methods sometimes referred to as *line search* [13].

We use the following procedure in the experiments in this paper [8]. From an initial point in parameter space, a search along each individual parameter co-ordinate axis is performed (all other parameters are taken as fixed). The value of the training set NDCG is computed for each of N sample points along the line, centred on an initial point in parameter space. The location of the best NDCG value is recorded. The distances of these optima from the initial point define a "promising" direction in parameter space. For example, if a distant optimum was found for a particular dimension, then there would be a large component of this promising direction in that dimension. Finally, we sample NDCG along this promising direction, generally involving changes now in all parameters. If we find an NDCG value that is better than where we started, we move the current estimate of the best parameter vector.

We define an *epoch* as one cycle through all parameters, plus the final search along the promising direction. Thus if there are P parameters then we perform $P + 1$ line search operations per epoch.

At the end of each epoch we reduce the scale over which the samples are taken by a factor of 0.85. The starting point for the next epoch uses the chosen optimum of the previous epoch.

We stop training if either we have reached a (rather arbitrary) limit of 24 epochs or we have seen 3 successive epochs with no improvement.

While this procedure is unquestionably computationally expensive, this particular kind of line search is at least parallelizable, as each co-ordinate axis can be searched independently. Another common technique we could use in the future to speed up the line search is called search by golden section [13], where the optimum is bracketed by ever reducing intervals.

2.3 Substitute Measures

Some approaches to optimisation formulate the problem with an explicit or implicit objective function (that which the method tries to optimise) that may be different from any traditional measure of retrieval effectiveness. For example, approaches based on linear regression typically use a least-squares error function in the predicted variable. Logistic regression attempts to predict a binary categorisation, using its own error function. Other categorisation approaches similarly make use of particular error functions which may be very different from any recognisable IR measure.

It may be observed that the general recommendation in the machine learning literature (see for example [12]) is that the objective function should indeed be the function one really wants to optimise. As against this, it may be that the measures we really care about are not susceptible to suitable methods. Clearly, if we pursue other measures purely because we have suitable optimisation methods to go with them, then we need to pay great attention to the question of whether good parameters according to the substitute measures are also good for the measures we care about.

Logistic regression is an example where the objective of the optimiser is seen as optimising the prediction of relevance itself, rather than a resulting effectiveness measure such as MAP or NDCG. The assumption is that if we can predict the relevance of each document well, we can also do well on the regular IR measures. An alternative approach is to consider pairs of documents rather than single documents. That is, what we want to optimise is a ranking, or a measure based on rankings, and a simple test of a ranking is whether a given pair of documents is in the right order. This approach has some support in the IR literature (with the bpref measure of [3]) and in the machine learning literature (with the learner of [6]). The bpref work indicates that the choice of pairs is critical for the IR case; this relates to the fact that measures such as MAP or NDCG are very heavily weighted to the top end of the ranking.

In this paper, we use a differentiable cost measure based on the scores of pairs of documents. We discuss this measure in the next section.

2.4 RankNet

In RankNet [4], a cost function (referred to here as RNC) is defined on the *difference in scores* of a pair of documents. The training data is defined as some set of pairs of documents on which the relevance judgements give us a preferred order for a particular query. Thus a pair consisting of one relevant and one non-relevant, or one highly relevant and one minimally relevant, might be a candidate. The choice of pairs used in RankNet is discussed further in section 3.1.

The basic idea behind RankNet is as follows. The algorithm assumes that it is provided with a training set consisting of pairs of documents d_2, d_1 together with a target probability $\bar{P}_{12}$ that document d_1 is to be ranked higher than document d_2. They define a ranking function (model) of the form $g : R^D \mapsto R$, in other words, mapping a vector of D feature values to the reals. The rank order of a set of documents is therefore defined by the values taken by g. In particular, it is assumed that $g(d_1) > g(d_2)$ means that the model ranks d_1 higher than d_2.

The map from the outputs g to probabilities is modelled using a logistic function $P_{12} \equiv e^{-Y}/(1+e^{-Y})$ where $Y \equiv g(d_2) - g(d_1)$, and P_{12} is the probability that d_1 is ranked higher than d_2. Burges et. al. then invoke the cross-entropy error function to penalize pair ordering errors:

$$C(Y) = -\bar{P}_{12} \log P_{12} - (1 - \bar{P}_{12}) \log(1 - P_{12}). \qquad (1)$$

This is a very general cost function allowing us to use any available uncertainty we may have concerning the pairwise ratings. In this paper, we take the pair ordering data as certain, and so for us, $\bar{P}_{12}$ are always one. With this simplification, the RankNet cost becomes:

$$\mathrm{RNC}(Y) = \log\left(1 + e^Y\right) \qquad (2)$$

where Y is the difference in the scores of the two documents in the pair, positive if the documents are in the wrong order.

The RankNet cost function assigns a cost to a pair in the wrong order, increasing as the score difference increases in that direction. It does not go immediately to zero, but asymptotically approaches zero, as the score difference increases in the other direction. Thus the gradient of RNC not only encourages the pair to be in the right order, but encourages them to have at least some separation in their scores.

Given a scoring/ranking function g that is differentiable with respect to the parameters, RNC will also be differentiable with respect to the parameters. Thus we will be able to discover the gradient of RNC, and a gradient descent procedure for discovering an optimum is feasible.

The full RankNet method is based on a neural net. The only aspect we have taken from this work is RNC itself, together with the idea of gradient descent.

Gradient descent on RNC

Our approach to this is a special case of a textbook implementation of the back-propagation algorithm [11]. From a starting point (set of parameter values), the chosen pairs are evaluated in a random order. An epoch is a cycle through all pairs. For each pair, its contribution to the gradient vector for RNC is evaluated. A small step (determined by a training rate) is taken in parameter space, in the direction of reducing cost. The initial training rate is 0.001, and this is reduced every time an epoch results in an increase in the total RNC. As with the line search optimization procedure described in section 2.2, we do a maximum of 24 epochs.

Ranking functions

The various ranking functions used are defined in more detail in section 3.2. In general, they involve a single strong content feature, namely BM25F, combined with a number of additional weak features (both static and dynamic) provided by a commercial search engine. The parameters to be optimised are (a) the linear weights of the combination of BM25F with the other features, and (b) the highly non-linear parameters of BM25F itself. That is, BM25F is treated as a single feature in a single-layer neural net with the other features, but the back-propagation is extended to the internal parameters of BM25F.[1]

The derivatives of RNC with respect to the linear weights are simple, but those with respect to the BM25F parameters are more complex – they are presented in the Appendix. The back-propagation procedure extends to these parameters. This aspect is an original contribution of the present paper.

2.5 Train, validate, test

Given a fixed set of queries with associated relevance evaluations, we will frequently want to hold out some set of queries from the training process in order to use them for testing – we wish to avoid overfitting the training set, and such a held-out test set will help us establish the generalisability of our trained model. However, a common procedure in machine learning training methods is to do a 3-way division of the evaluated material, into training, validation and testing. The validation step has several possible functions. Conventionally, one use is to avoid overfitting, as follows. We go through an iterative training process on the training set, and at each iteration evaluate on the validation set. When we reach some stopping point, we would normally expect the best value of the objective function in the training set to be that achieved in the last iteration. However, we choose the model on the basis of the validation set results, which often means choosing an earlier iteration. This heuristic is found to work well.

RankNet's Validation Trick

In [4] the validation stage has a dual purpose. The training process is driven by gradient descent on the RNC. However, instead of validating on RNC, the authors validate by choosing the best model (over training epochs) according to *validation set NDCG*, the effectiveness measure we are really interested in. So in addition to preventing us from overfitting, it also serves to reject models that were only good according to RNC by checking them on the true metric of interest.

3. EXPERIMENTS

Given a parameterised ranking function, training set and a measure that we wish to optimise, we have to choose a training (optimisation) method. A perfect training method would find the true global optimum of the desired measure in the full parameter space; however, any realistic method is likely to fail in this task, and result in a set of parameters which is not exactly optimal. We are concerned with the extent to which we may fall below optimal performance. In

[1] We are currently also working on 2-layer nets, which need more data but may perform better than linear models [4]

the case of a heuristic line-search, such sub-optimal performance will be purely because the line search fails to find the true global optimum. In the case of gradient descent based on RNC, we have the additional factor that optimal cost is not necessarily the same as optimal effectiveness in terms of our real objective.

As we have seen, the choice of training method may depend on the number of parameters in the model. Another variable that may be important is the size of the training set. We might expect that we need a larger training set if we have (want to make effective use of) more parameters. Also, while all training methods might be expected to do better with larger training sets, the relationship between performance of the method and training set size might vary between trainers.

The variables that we wished to consider in the experiments described below are as follows:

- The number of parameters in the ranking function;
- The size of the training set (number of queries);
- The use of a gradient descent method based on RNC, versus a form of line search.

Our experiments involve training sets ranging from 16 to 2048 rated queries[2] (section 3.1) with rankers with 2 or 9 or 375 tunable parameters (section 3.2).

3.1 Data

Because we wished to explore these variables (specifically the first two) in regions which are not commonly available in public data sets, we have conducted our experiments on data sets taken from a commercial search engine. These data sets provide us with a large number of queries with relevance judgements, and also allow us to use many distinct features which may be combined in weighted fashion. Thus the feature weights provide us with most of our many parameters.

The data sets are based on a set of English queries (numbers given below) sampled from query logs on a large web search engine. For each query, we select a candidate document collection consisting of approximately 2500 top ranked documents, of which approximately 30 are rated by judges on a 5 point relevance scale (including non-relevant). The remainder are labelled as not relevant. In addition, there are a random sample of a few hundred unretrieved documents from the set of documents containing all query terms, which represent documents that are poor matches with high probability. However, idf values for the term weights are taken from the original collection.

To facilitate the investigation of the effect of training set size, we vary the size of the training set, starting at 16 queries, and increasing by steps of $\sqrt{2}$ to 2048. For each training set, there is an associated validation set of one-quarter the size. The test set is a fixed set of 512 queries. For validation and test, NDCG is evaluated from the full ranking; unrated documents are treated as non-relevant.

The remainder of this section describes how the choice of optimisation method affects the *training* data that we use in our experiments.

Line search on NDCG

Line search training can use all the documents that we have for each query, both rated and un-rated, giving approximately 2500 documents per query. We assume that unrated documents get zero gain in the NDCG computation.

Gradient Descent on RNC

If each unlabeled document were used to compute the set of all pairs for the computation of RNC, there would be an extremely large number of pairs per query. Also the evidence from the bpref study [3] suggests that we need to be more selective about the pairs we use. To reduce the number of pairs to a more manageable size, we adopted a similar sampling strategy to that described in [4] as follows. We randomly subsampled from the unrated documents to give a subset of the same size as the average number of rated documents per query. These documents are treated as irrelevant. We do not use tied pairs.

As argued in [4], it is possible that this is not just good for computational reasons. In performing this subsampling process, we are substantially increasing the probability that any unrated document that we use in our training process really is irrelevant, and therefore should get zero gain in NDCG.

3.2 Ranking functions

This section describes the three ranking models we shall investigate.

The 2-parameter model

The *2-parameter* model is simple BM25, the parameters being the usual BM25 parameters of k controlling term frequency saturation rate and b controlling document length normalisation. Note that $k \geq 0$ and $0 \leq b \leq 1$. The final document score is the sum of BM25 with a single fixed weight (parameter-free) query independent feature. This query independent feature captures the quality of the document according to the link structure of the web, independent of query and content.

The 9-parameter model

The *9-parameter* model is BM25F [14, 15] (see appendix), this time in a trainable combination with the single query independent feature used in the 2-parameter model. The BM25F ranking function is an extension of the BM25 function that admits extra degrees of freedom to capture the relative semantic significance of any *fields* that may be contained in the document.

In our current corpus, there are four fields: incoming anchor text, URL, title and body. Each field has its own b and its own term-frequency weight w. Again, $0 \leq b \leq 1$ and $w \geq 0$. Since there is redundancy between varying the weights and varying the k value [14], we choose to fix the single k value and vary the four weights. The final scoring function is BM25F linearly combined with the static feature (one more weight). The formula for this ranking function and the derivatives that may be used in back-propagation are given in the appendix.

We note that the 9-parameter model is very similar to the model used in our successful submission to the TREC Web Track in 2004 [15]. It thus represents something close to the state of the art of web search at TREC.

[2] We are currently working on experiments on training sets upto 10 times larger.

The 375-parameter model

The *375-parameter* model uses a large number of weak features. These features are not described in detail here; the object is to illustrate the process. As with the 9-parameter model, the scoring function first calculates BM25F and then combines the BM25F score in a weighted linear fashion with the remaining features.

4. RESULTS

We investigate the relevance performance of the three rankers described in the last section over a range of training set sizes.

4.1 The 2-parameter ranker

With two parameters, we can plot an approximate effectiveness surface by running a full grid over the parameter space. In Figure 1 we see three surfaces, based on the 64-query training set and the 512-query test set, in the form of contours. The three are: RNC on the selected pairs from the training set; NDCG as calculated on the training set; and NDCG as calculated on the test set.

The striking thing about this figure is the similarity between Figure 1a, the RNC computed from rated and random sub-sampled documents from only 64 queries, and Figure 1c, the target test set NDCG computed from all available documents from 512 queries. Both these surfaces appear smooth (good for search- and gradient-based optimisation alike) and seem to have very similar structure. In contrast, the training set NDCG surface in Figure 1b appears bumpy. This gives us at least an intuitive feel for why direct optimisation of the training set NDCG may give unpredictable results for small training sets.

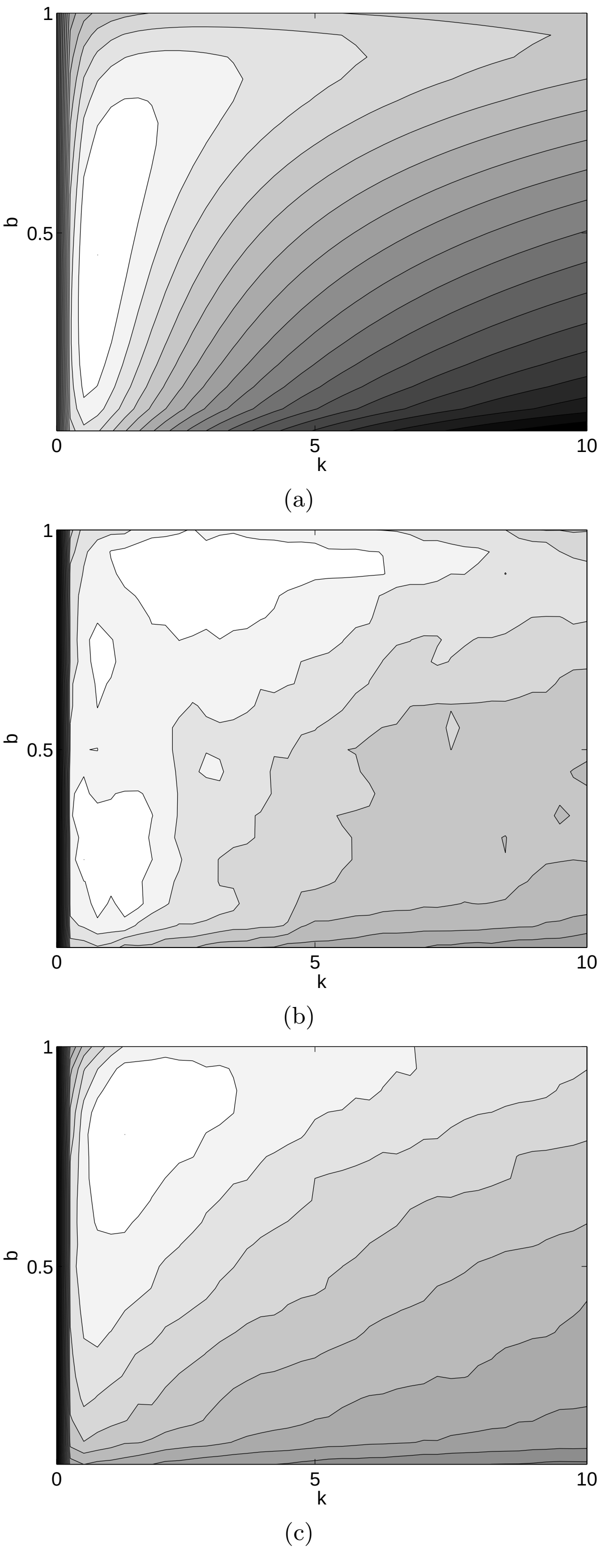
Armed with the derivatives of the RNC with respect to k, b (see appendix), we ran line search on NDCG verses gradient descent optimisation on RNC over the range of training query set sizes. We initialised both at $k = 1.0$ and $b = 0.5$. Our principle result here is that that gradient descent and line search performed almost identically over all query set sizes. In all cases, the trained ranker gave a test set NDCG of about 0.30 ± 0.02[3]

In addition, we observed that the test set NDCG did not improve significantly from the initial starting point. In other words, we would have got similar test set NDCG performance with the 2-parameter ranker set to its initial point, with no optimization of BM25 parameters.

4.2 The 9-parameter ranker

As we increase the number of parameters, the 'ground truth' represented by the grid search is not accessible to us. We use the line search procedure described in section 2.2 as a substitute baseline. Because it directly optimizes NDCG, there is reason to believe that it should perform well against the gradient descent approach.

With this ranker, we again compared the performance of line search against gradient descent on the RNC, using derivatives of RNC with respect to BM25F parameters presented in the appendix. We set identical initial BM25F weights as $w_s = 1.0$, $b_s = 0.5$ (see appendix) and the initial



(a)

(b)

(c)

Figure 1: Objective function contours for training and test sets for the 2-parameter model: (a) RNC in the training set; (b) NDCG in the training set; (c) NDCG in the test set. (Training set of 64 queries. RNC in the training set based on chosen pairs only; Both NDCG plots, train and test, are based on all documents.)

[3] This confidence limit is 95% on true population NDCG value given the 512 sample size, and unless otherwise stated, applies to all our results on the 512 query test set.

weight of the query independent feature to be the same as it was for the 2-parameter model.

The results are shown in Figure 2 as “Line search 9” and “Gradient descent 9”. We can see that the two optimization techniques are comparable in performance, with gradient descent being slightly more consistent.

We also report a significant improvement in performance over the initial model (epoch zero), thus proving the value of BM25F tuning in this ranker. The initial model had a test set NDCG of 0.27. As you can see from Figure 2, both tuning mechanisms beat this by a very significant margin for all but the smallest training set sizes. We conclude that the extra degrees of freedom in a tuned BM25F model are a valuable addition to the parameter space.

4.3 The 375-parameter ranker

In this scenario the line search baseline is not feasible using commonly available computer hardware. Therefore we set out to compare the use of gradient descent on *all* the parameters (“Gradient descent 375” in Figure 2) with the use of gradient descent on the BM25F parameters alone and line search on the rest (“Line search 375”).

As you would expect, such a large model needs a certain amount of training data before it starts performing well. Paired t-tests indicate that the 375-parameter model significantly outperforms the 9-parameter model for training sets of 256 queries or more. It is also significantly worse for training sets of 45 queries or fewer. All these differences were significant at $p < 0.01$ (and usually at $p < 0.001$). We also measured changes as the collection size grows, showing significant improvements in the 375-parameter model, but no significant changes in the 9-parameter model (Table 1).

We note that this result indicates that many weak features can add significantly to a state-of-the-art TREC Web Track ranker.

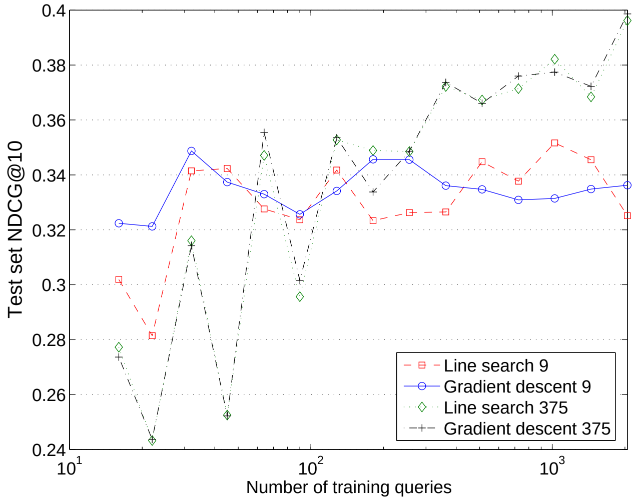


Figure 2: NDCG on the test set for different training set sizes, different numbers of parameters per model, and the two optimisation methods.

Effect of BM25F tuning

Assuming that we wish to adopt the gradient descent approach to tuning BM25F parameters, it is natural to consider how much improvement the tuning process makes over the initialisation point. In Figure 3 we repeat the “Gradient descent 375” run, now labelled “Tuned BM25F”. We show three further runs: the first is the test NDCG for the model without BM25F (with 367 parameters, “No BM25F”), the second is a run with the BM25F model with its initial settings (“Untuned BM25F”) and the third is the line-search tuned BM25F parameters from the corresponding “Line search 9” runs as *fixed* values in the 375-parameter ranker (“Precooked BM25F”).

We see that the ranker without BM25F is consistently beaten for training sets above about 100 queries. Adding even an untuned BM25F gives a statistically significant improvement for training set sizes greater than or equal to 512 queries. However, the difference between the remaining three models is not statistically significant. It is surprising that tuning BM25F in this context makes so little difference. We are currently working on larger query sets (training and test) to see under what conditions, if any, this difference becomes significant.

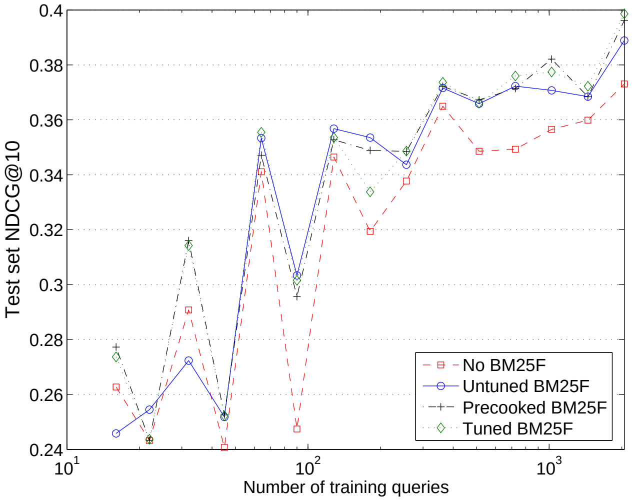


Figure 3: The effect of tuning BM25F within a large ranking model. How NDCG improves by adding an untuned BM25F feature, and how it does not change much by tuning it in the presence of hundreds of other features.

5. CONCLUSIONS

We may summarise our results as follows:

- We have started to build up some convincing evidence that gradient on RNC can be used effectively to tune the parameters of ranking functions such as BM25F. This makes training of such traditional ranking functions much less computationally expensive.
- Gradient descent and line search track each other quite closely (in final model effectiveness if not in selected parameter values).

- Final model effectiveness is somewhat unstable when we have only a small training set. This result is particularly true if we have a large number of parameters.

- A large number of features, each with its own linear weight, may be used to improve performance *provided that* we have a large enough training set.

- BM25F tuning seems to become less critical with large feature sets.

We find the relative success of gradient descent, using RNC, very interesting. This is a far more efficient method than line search when we work with subsampled pairs as discussed, and seems to do just as well, despite the fact that it uses the 'wrong' objective function. The RankNet procedure seems to be a pretty robust process. We do not of course know how close we are to any 'true' optimum, but it appears at least difficult to do better.

It remains the case that optimising these ranking functions is an intuitive process involving many heuristics. Very many decisions were taken in setting up both these of experiments, any one of which could be affecting the results we have obtained. One example is the train-validation split – for situations where the potential number of rated queries is restricted, holding out queries for validation (and/or test) limits the training set in ways that might be bad for training. Similarly the use of judge effort for deep pool evaluation on smaller numbers of queries or shallow pool on larger numbers is a significant issue. A few of these variables have been explored in a limited way during the preparation of this paper, but there remain many unexplored areas. Nevertheless, the apparent robustness of gradient descent is encouraging.

	Number of training queries		
	64	362	2048
LS9	0.328	-0.001 (p=0.730)	-0.002 (p=0.564)
GD9	0.333	+0.003 (p=0.563)	+0.003 (p=0.533)
GD375	0.355	+0.018 (p=0.013)	+0.043 (p<0.001)

Table 1: Adding more training queries does not give significant improvements for the 9 parameter models, but does for the 375 parameter model.

6. ACKNOWLEDGEMENTS

We would like to thank Andy Laucius and Timo Burkard for both providing the data and many useful discussions. Also thanks to Greg Hullender and Erik Selberg for helpful feedback on an earlier draft.

7. REFERENCES

[1] C. M. Bishop. *Neural networks for pattern recognition.* Oxford University Press, Oxford, UK, 1996.

[2] C. Buckley and G. Salton. Optimization of relevance feedback weights. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 351–357, New York, 1995. ACM Press.

[3] C. Buckley and E. Voorhees. Retrieval evaluation with incomplete information. In K. Järvelin, J. Allan, P. Bruza, and M. Sanderson, editors, *SIGIR 2004: Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 25–32, New York, 2004. ACM Press.

[4] C. J. C. Burges, T. Shaked, E. Renshaw, et al. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, Bonn, 2005.

[5] J. Gao, H. Qi, X. Xia, and J.-Y. Nie. Linear discriminant model for information retrieval. In G. Marchionini, A. Moffat, J. Tait, R. Baeza-Yates, and N. Ziviani, editors, *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 290–297, New York, 2005. ACM Press.

[6] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. MIT Press, 2000.

[7] D. A. Hull, J. O. Pedersen, and H. Schütze. Method combination for document filtering. In H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, editors, *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 279–287, New York, 1996. ACM Press.

[8] G. Hullender. Personal communication, 2004.

[9] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In N. J. Belkin, P. Ingwersen, and M.-K. Leong, editors, *SIGIR 2000: Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 41–48, New York, 2000. ACM Press.

[10] T. Joachims. A support vector method for multivariate performance measures. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 377–384, New York, NY, USA, 2005. ACM Press.

[11] Y. LeCun, L. Bottou, G. B. Orr, and K. R. Müller. *Efficient backprop in neural networks: tricks of the trade.* Springer, 1998.

[12] D. D. Lewis. Evaluating and optimizing autonomous text classification systems. In E. A. Fox, P. Ingwersen, and R. Fidel, editors, *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 246–254, New York, 1995. ACM Press.

[13] D. G. Luenberger. *Linear and nonlinear programming.* Addison Wesley, 1984.

[14] S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In D. A. Evans, L. Gravano, O. Hertzog, C. X. Zhai, and M. Ronthaler, editors, *CIKM 2004: Proceedings of the 13th ACM Conference on Information and Knowledge Management*, pages 42–49, New York, 2004. ACM Press.

[15] H. Zaragoza, N. Craswell, M. Taylor, S. Saria, and

S. Robertson. Microsoft Cambridge at TREC 2004: Web and HARD track. In E. M. Voorhees and L. P. Buckland, editors, *The Thirteenth Text REtrieval Conference, TREC 2004*, NIST Special Publication 500-261. Gaithersburg, MD: NIST, 2005.

APPENDIX

A. DERIVATIVES OF RNC WRT BM25F

In this appendix, we derive the derivatives of the RNC with respect to the BM25F parameters. We define the BM25F ranking function as:

$$F = \sum_t I_t \tau\left(f_t\right) \tag{3}$$

where I_t is the Robertson-Spark-Jones term weight (idf), and τ is some sort of saturation function and f_t is the field aggregated term frequency. We will use the following usual saturation function:

$$\tau(f) = \frac{f}{k+f}, \tag{4}$$

and the definition of f_t is:

$$f_t = \sum_s \frac{w_s f_{ts}}{\beta_s} \tag{5}$$

where w_s is the field weight, and the document length normalisation model is defined in the usual way, per field, as:

$$\beta_s = 1 - b_s + b_s l_s / \bar{l}_s \tag{6}$$

We first derive some general results, and then go on to use these for the required derivatives with respect to k, w_s, b_s.

A.1 Derivatives of BM25F

With respect to k

$$\begin{aligned} \frac{\partial F}{\partial k} &= \sum_t I_t \frac{\partial \tau}{\partial k} \\ \frac{\partial F}{\partial k} &= -\sum_t \frac{I_t f_t}{(k+f_t)^2} \end{aligned} \tag{7}$$

With respect to w_s

$$\frac{\partial F}{\partial w_s} = \sum_t \frac{I_t k f_{ts}}{\beta_s (k+f_t)^2} \tag{8}$$

With respect to b_s

$$\frac{\partial F}{\partial b_s} = \sum_t I_t \frac{\partial \tau}{\partial b_s} = \sum_t I_t \frac{\partial \tau}{\partial f_t} \frac{\partial f_t}{\partial b_s} \tag{9}$$

giving:

$$\frac{\partial F}{\partial b_s} = \sum_t \frac{I_t k w_s f_{ts} \left(1 - l/\bar{l}_s\right)}{(k+f_t)^2 \beta_s^2} \tag{10}$$

A.2 RankNet Cost Derivatives

This section uses the partial derivatives of BM25F to get the required derivatives of the RNC, here denoted C. Let β be a generic BM25F parameter, $y^{(1)}, y^{(2)}$ be a pair of scores where d_1 has a better rating than d_2 and $Y \equiv y^{(2)} - y^{(1)}$:

$$\frac{\partial C}{\partial \beta} = \frac{\partial C}{\partial Y} \frac{\partial Y}{\partial \beta} = \frac{\partial C}{\partial Y} \left(\frac{\partial y^{(2)}}{\partial \beta} - \frac{\partial y^{(1)}}{\partial \beta} \right) \tag{11}$$

With respect to score differences

$$C(Y) = \log\left(1 + e^Y\right) \tag{12}$$

$$\frac{\partial C}{\partial Y} \equiv C'(Y) = \frac{e^Y}{1+e^Y} \tag{13}$$

A.3 With respect to BM25F parameters

We assume that the scoring function combines N features linearly, and that the Nth feature is BM25F. We have

$$y^{(k)} = \sum_{i=0}^{N} w_i x_i^{(k)} \tag{14}$$

and so:

$$\frac{\partial y^{(k)}}{\partial \beta} = w_N \frac{\partial F^{(k)}}{\partial \beta} \tag{15}$$

where $F^{(k)}$ is BM25F. Now the gradient we require is

$$\frac{\partial C}{\partial \beta} = \frac{\partial C}{\partial Y} w_N \left(\frac{\partial F^{(2)}}{\partial \beta} - \frac{\partial F^{(1)}}{\partial \beta} \right) \tag{16}$$

The required partial derivatives of the RankNet cost with respect to k, w_s and b_s can be obtained by substituting (7), (8) and (10) respectively for $\partial F^{(k)}/\partial \beta$ in (16), and $\partial C/\partial Y$ is given by (13).