SoftRank with Gaussian Processes

Edward Snelson
Microsoft Research, Cambridge, UK
esnelson@microsoft.com

John Guiver
Microsoft Research, Cambridge, UK
joguiver@microsoft.com

Abstract

We address the problem of learning to rank based on a large feature set and a training set of judged documents for given queries. Recently there has been interest in using IR evaluation metrics to assist in training ranking functions. However, direct optimization of an IR metric such as NDCG with respect to model parameters is difficult because such a metric is non-smooth with respect to document scores. Recently Taylor et al. presented a method called SoftRank which smooths a metric such as NDCG by introducing uncertainty into the scores, thus making it amenable for optimization. In this paper we extend SoftRank by combining it with a Gaussian process (GP) model for the ranking function. The advantage is that the SoftRank smoothing uncertainties are naturally supplied by the GP, reflecting the underlying modelling uncertainty in individual document scores. We can also use these document uncertainties to rank differently, depending on how risky or conservative we want to make the ranking. We test our method on the publicly available LETOR OHSUMED data set and show very competitive results.

1 IR metrics

Our task in information retrieval is to choose and present, possibly in an ordered list, a set of documents relevant to the query entered by a user. In order to design and improve retrieval systems we need to evaluate the quality of the retrieved results. In practice this is usually done using a set of judged documents for multiple queries. The documents are judged for relevance to the query either using binary labels or a graded scale. An IR metric or utility is a function of the judged labels for a returned set of documents. Many such metrics have been developed (see e.g. [1]) to try to capture various aspects of user preference for the retrieved set. For example, in a ranked list a user presumably pays most attention to the head of the list, and we therefore want to make sure we get the very relevant documents right at the top.

1.1 NDCG

Normalized Discounted Cumulative Gain (NDCG) [2] is an IR metric that is a function of graded relevance labels, typically 0 (bad) to 4 (perfect), which focuses on the top of a ranking using a discount function. It is defined as :

$$G_R = G_{R,\max}^{-1} \sum_{r=0}^{R-1} g(r) D(r) \tag{1}$$

where the gain $g(r)$ of the document at rank r is usually an exponential function $g(r) = 2^{l(r)}$ of the labels $l(r)$ (or rating) of the document at rank r. R is the truncation rank. Where no subscript is defined, it should be assumed that $R = N$, the number of documents associated with the query under consideration. A popular choice for the rank discount is $D(r) = 1/\log(2 + r)$ and $G_{R,\max}$ is the maximum value of $\sum_{r=0}^{R-1} g(r)D(r)$ obtained when the documents are optimally ordered by decreasing label value. The intuition behind the NDCG metric is as follows. The discount function ensures we focus on getting documents in the right order at the top of the ranking. The gain function ensures we place more emphasis on the highly relevant documents. The normalization allows us to meaningfully average over many different queries, for which there may be some queries with many more relevant documents than others.

2 Learning to rank

To produce a ranked list of documents for a given query, many methods use a score function to map from document (and query) features $\mathbf{x}$ to a real valued score s. For a document indexed by j:

$$s_j = f(\mathbf{x}_j, \mathbf{w}) \ , \tag{2}$$

where $\mathbf{w}$ is a vector of parameters defining the score function. To produce a ranking the documents are ordered according to score. An IR metric such as NDCG is then used to evaluate the ranked list.

The learning task is then to find suitable parameters $\mathbf{w}$ of the score function, from a training set of judged query-document pairs. One way of doing this is to ignore the IR metric completely for the learning, and to concentrate on the labels themselves. For example, we can perform regression directly on the (graded) labels, or perhaps more properly, ordinal regression [3]. An alternative way of training is to first turn the labels into pairwise preferences [4, 5, 6]. However, given we are to be evaluated on a particular IR metric, it may be advantageous to use the same metric to guide the training of the parameters of the ranking function. The successful LambdaRank is an example of a system which implicitly does this [7]. The most direct way to achieve this is to optimize the IR metric on the training data with respect to the parameters of the ranking function. Efficient optimization in a high dimensional parameter space can only really be achieved using gradient based methods. Here we hit a problem: typical IR metrics such as NDCG are non-smooth functions of the document scores s_j (and hence parameters $\mathbf{w}$) because they only depend on the ordering of the scores rather than the score values themselves. Such metrics are therefore not amenable to gradient based optimization.

3 SoftRank

Recently Taylor et al. [8] have developed a general method called SoftRank to smooth IR metrics, giving differentiable objective functions suitable for gradient optimization. The high level idea is to replace the deterministic score of (2) with an uncertain or probabilistic one:

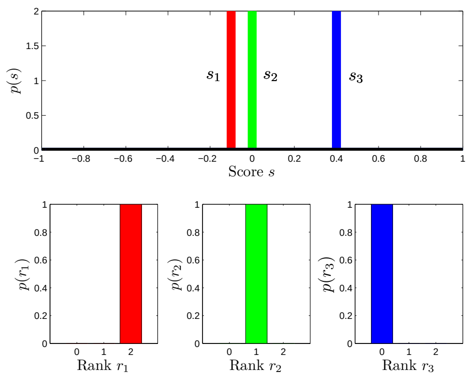
$$p(s_j) = \mathcal{N}(s_j|\bar{s}_j, \sigma_s^2) = \mathcal{N}(s_j|f(\mathbf{x}_j, \mathbf{w}), \sigma_s^2) \ , \tag{3}$$

where $\mathcal{N}$ denotes a Gaussian distribution and σ_s^2 is a shared smoothing variance. The softened objective function is then created by computing the *expectation* of the original IR metric with respect to these score distributions. An exact analytical form for this expectation is intractable in general, but a close approximation can be made by first converting from score to rank distributions. We give here a high level exposition of this procedure; for more details refer to [8].

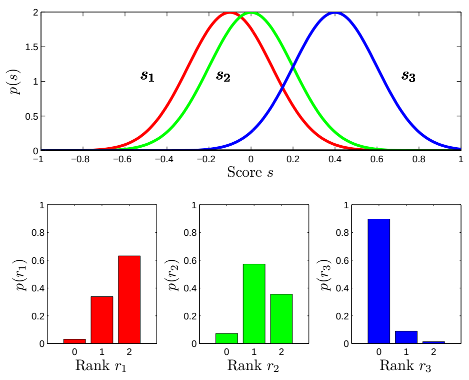
3.1 From score to rank distributions

A set of N score distributions (3) defines a distribution on the possible $N!$ orderings of the documents. For example, to sample an ordering from the distribution we would first sample N scores from the score distributions, and then sort the scores to obtain the ordering. The first point to note is that any procedure we have for computing expectations involving such a sort will not be differentiable, so we have to avoid this. The second point is that in many cases (e.g. with NDCG) we do not actually need the full ordering distribution to compute the expectation, and instead we can deal with individual document rank distributions $p_j(r)$. This is the probability that document j will be at rank r under the N score distributions. For illustration refer to Figure 1. Figure 1a shows deterministic scores for three documents and their corresponding hard rank distributions. Figure 1b shows three Gaussian score distributions and their corresponding soft rank distributions.

For the purposes of this paper it is sufficient to know that we can compute close approximations to the rank distributions $p_j(r)$ using a recursive procedure that does not involve an explicit sort, and hence we can compute derivatives with respect to parameters. The computation involves as an



(a) Three document scores as point values and their corresponding rank distributions.



(b) Smoothed scores for 3 documents and the resulting 3 rank distributions.

Figure 1: From score to rank distributions

intermediate step the calculation of the pairwise probabilities π_{ij} that document i ranks higher than document j:

$$\pi_{ij} \equiv \Pr(S_i - S_j > 0) = \int_0^\infty \mathcal{N}(s|\bar{s}_i - \bar{s}_j, 2\sigma_s^2)ds \; . \tag{4}$$

3.2 SoftNDCG

As an example of smoothing an IR metric we focus on NDCG. We can rewrite the formula for NDCG (1) as a sum over document indices rather than document ranks:

$$G = G_{\text{max}}^{-1} \sum_{j=1}^{N} g(j) D(r_j) \; . \tag{5}$$

Now the SoftNDCG is defined as the expected NDCG with respect to the rank distributions:

$$\begin{aligned} G_{\text{soft}} = E[G] &= G_{\text{max}}^{-1} \sum_{j=1}^{N} g(j) E[D(r_j)] \\ &= G_{\text{max}}^{-1} \sum_{j=1}^{N} g(j) \sum_{r=0}^{N-1} D(r) p_j(r) \; . \end{aligned} \tag{6}$$

The additive separation of NDCG over documents in equation (5) makes it clear why we actually only need the marginal rank distributions $p_j(r)$ rather than the full distribution over all rankings.

Finally, to optimize G_{soft} we need to compute derivatives with respect to the parameters $\mathbf{w}$ of the ranking function: $\frac{\partial G_{\text{soft}}}{\partial \mathbf{w}}$. This is essentially just application of the chain rule, noting that G_{soft} is a function of the rank distributions $p_j(r)$, which are in turn functions of the pairwise probabilities π_{ij}, which are in turn functions of the score means $\bar{s}_j$, which are finally functions of the ranking parameters $\mathbf{w}$: $\bar{s}_j = f(\mathbf{x}_j, \mathbf{w})$. Taylor et al. [8] use a 2-layer neural net for the ranking function f.

3.3 Limitations of SoftRank

Taylor et al. [8] demonstrated good results using SoftRank, showing that optimizing SoftNDCG on the training set yields test set results comparable to the state-of-the-art LambdaRank [7], but having several additional benefits, including the ability to explicitly train any metric which can be built from score probabilities. However, conceptually at least, some problems still remain. What is the meaning of the smoothing variance parameter σ_s^2, and how should it be set? In [8] it is fixed during training, although an extra degree of freedom (the scale of f) means that it is effectively being optimized. From a Bayesian perspective though it makes sense to regard the variance σ_s^2 as the uncertainty associated to the function due to uncertainty in the model parameters $\mathbf{w}$. In this sense there should be an individual posterior uncertainty for each document σ_j^2, reflecting the fact that we will be more confident about some documents than others. In practice it is hard to achieve this tractably using a neural net.[1] Gaussian process models, however, are non-linear models which do give analytically tractable uncertainty estimates. We turn to this class of models in the next section.

4 Gaussian process (GP) regression models

We briefly summarize GPs for regression, but see e.g. [9] for more detail. A Gaussian process defines a prior distribution over functions $f(\mathbf{x})$, such that any finite subset of function values $\mathbf{f} = \{f_n\}_{n=1}^N$ is multivariate Gaussian distributed given the corresponding feature vectors $\mathbf{X} = \{\mathbf{x}_n\}_{n=1}^N$:

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}\big(\mathbf{f}|\mathbf{0}, K(\mathbf{X},\mathbf{X})\big) \ . \tag{7}$$

The covariance matrix $K(\mathbf{X},\mathbf{X})$ is constructed by evaluating a covariance or kernel function between all pairs of feature vectors: $K(\mathbf{X},\mathbf{X})_{ij} = K(\mathbf{x}_i,\mathbf{x}_j)$.

The covariance function $K(\mathbf{x},\mathbf{x}')$ expresses some general properties of the functions $f(\mathbf{x})$ such as their smoothness, scale etc. It is usually a function of a number of hyperparameters θ which control aspects of these properties. A standard choice is the ARD + Linear kernel[2]:

$$K(\mathbf{x}_i,\mathbf{x}_j) = c \exp\left[-\sum_{d=1}^{D} \frac{\big(x_i^{(d)} - x_j^{(d)}\big)^2}{2\lambda_d^2}\right] + \sum_{d=1}^{D} w_d \, x_i^{(d)} x_j^{(d)} + w_0 \ , \tag{8}$$

where $\theta = \{c, \lambda_1, \ldots, \lambda_D, w_0, \ldots, w_D\}$. This kernel allows for smoothly varying functions with linear trends. There is an individual lengthscale hyperparameter λ_d for each input dimension, allowing each feature to have a differing effect on the regression.

In standard GP regression the actual observations $\mathbf{y} = \{y_n\}_{n=1}^N$ are assumed to lie with Gaussian noise around the underlying function: $p(y_n|f_n) = \mathcal{N}(y_n|f_n,\sigma^2)$. Integrating out the latent function values we obtain the marginal likelihood:

$$p(\mathbf{y}|\mathbf{X},\theta,\sigma^2) = \mathcal{N}(\mathbf{y}|\mathbf{0}, K(\mathbf{X},\mathbf{X}) + \sigma^2\mathbf{I}) \ . \tag{9}$$

which is typically used to train the GP by finding a (local) maximum with respect to the hyperparameters θ and noise variance σ^2.

Prediction is made by considering a new input point $\mathbf{x}$ and conditioning on the observed data and hyperparameters. The distribution of the output value at the new point is then:

$$\begin{aligned} p(y|\mathbf{x},\mathbf{X},\mathbf{y}) &= \mathcal{N}\big(y|\bar{s}(\mathbf{x}),\sigma^2(\mathbf{x})\big) \ , \\ \bar{s}(\mathbf{x}) &= K(\mathbf{x},\mathbf{X})\big[K(\mathbf{X},\mathbf{X}) + \sigma^2\mathbf{I}\big]^{-1}\mathbf{y} \\ \sigma^2(\mathbf{x}) &= K(\mathbf{x},\mathbf{x}) - K(\mathbf{x},\mathbf{X})\big[K(\mathbf{X},\mathbf{X}) + \sigma^2\mathbf{I}\big]^{-1}K(\mathbf{X},\mathbf{x}) + \sigma^2 \ , \end{aligned} \tag{10}$$

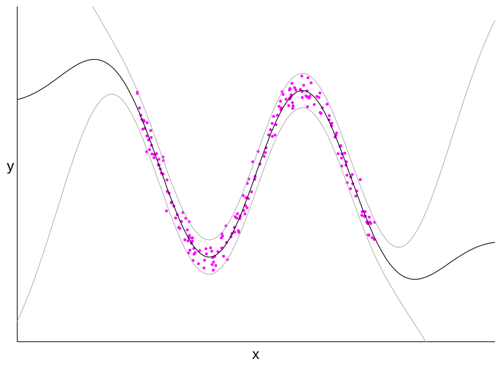
where $K(\mathbf{x},\mathbf{X})$ is the kernel evaluated between the new input point and the N training inputs. The GP is a non-parametric model, because the training data are explicitly required at test time in order to construct the predictive distribution.

Figure 2a shows the GP predictive mean and two standard deviation lines (10) plotted for a simple 1D noisy sine wave data set. The hyperparameters of the kernel have been set by optimizing the marginal likelihood as described above. The important point to notice is that the predictive variance captures the inherent uncertainty in the function, with tight error bars in regions of observed data, and with growing error bars away from observed data. It is this intuitive property we hope to introduce to the SoftRank model using a GP.

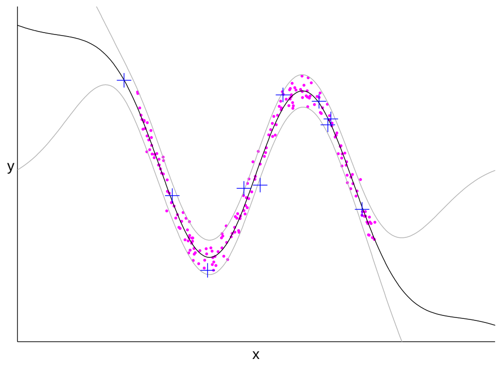
One limitation of GPs is that they are prohibitive for large data sets because training requires $\mathcal{O}(N^3)$ time due to the inversion of the covariance matrix. Once the inversion is done, prediction is $\mathcal{O}(N)$ for the predictive mean and $\mathcal{O}(N^2)$ for the predictive variance per new test case. We return to this problem in the next section.

[1] One would need to pass parameter distributions through the non-linearities of the neural net, and also define a proper likelihood, for a full Bayesian treatment.

[2] 'Automatic relevance determination' (ARD) [10]



(a) Full GP. Hyperparameters trained using marginal likelihood.



(b) GP based on 10 prototype points (marked as blue crosses). Prototypes and hyperparameters trained using regression objective.

Figure 2: Predictions for 1D noisy sine wave data. Predictive mean shown as black line, and 2 standard deviations shown as grey lines.

5 SoftRank with Gaussian processes

There are several ways in which Gaussian processes could be applied to ranking. The simplest is to ignore the NDCG objective for training, and to simply use a GP regression model on the labels themselves, treating them as real values. Indeed, we tried this first, but found it did not work especially well. The GP seemed to model the numerous 'bad' labels well, at the expense of the 'good' labels.[3] The direct optimization of NDCG in training with SoftRank was designed specially to get around this type of problem — the NDCG objective forces the model to concentrate on getting the relevant documents to the top of the ranking. We therefore wanted to combine a GP model with the SoftRank training scheme.

Notice that the GP predictive mean and variance functions (10) are of exactly the right form to be used as the score means and uncertainties in SoftRank (3): $p(s_j) = \mathcal{N}(s_j|\bar{s}_j, \sigma_j^2)$. Since we do not have a proper ranking likelihood (subject of future work), this combined GP/SoftRank model cannot be treated in a fully Bayesian way. Rather we regard the GP mean and variance functions (10) as parameterized functions to be optimized in the same way as the neural net in SoftRank. This treatment of the GP is similar to using a radial basis function (RBF) network in place of the neural net, except that we now have the addition of a variance function that exhibits the characteristic GP property of growing error bars away from observations.

Looking at equation (10) we see that the regression outputs $\mathbf{y}$ are now virtual or prototype observations — they are free parameters to be optimized. This is because all training information enters through the NDCG objective, rather than directly as regression labels. In fact we can go one step further and realise that the set of input vectors $\mathbf{X}$ on which the GP predictions are based do not have to correspond to the actual training inputs, but can be a much smaller set of free inputs. To summarize, the mean and variance for the score of document j are given by:

$$\begin{aligned} \bar{s}(\mathbf{x}_j) &= K(\mathbf{x}_j, \mathbf{X}^p)\left[K(\mathbf{X}^p, \mathbf{X}^p) + \sigma^2\mathbf{I}\right]^{-1}\mathbf{y}^p \\ \sigma^2(\mathbf{x}_j) &= K(\mathbf{x}_j, \mathbf{x}_j) - K(\mathbf{x}_j, \mathbf{X}^p)\left[K(\mathbf{X}^p, \mathbf{X}^p) + \sigma^2\mathbf{I}\right]^{-1}K(\mathbf{X}^p, \mathbf{x}_j) + \sigma^2 \ , \end{aligned} \tag{11}$$

where $(\mathbf{X}^p, \mathbf{y}^p)$ is a small set of M prototype feature-vector/score pairs. These prototype points are free parameters that are optimized along with the hyperparameters θ using the SoftNDCG gradient training.[4] The advantage of using a small set of M prototypes is that we now have a sparse model, and the previously problematic $\mathcal{O}(N^3)$ training time reduces to $\mathcal{O}(NM^2)$. This technique of optimizing a set of prototype points is in the same spirit as previously developed sparse GP methods

[3]Another approach would be to use a GP with an ordinal regression likelihood for the labels [11].

[4]The noise σ^2 could be omitted because it no longer corresponds to actual regression observation noise, but we left it in and optimized it because it helped with numerical stability.

such as the SPGP/FITC model [12], the main difference being that the SPGP optimization is done using the marginal likelihood rather than the SoftRank objective that we employ here.

To give an illustration of how this works we can actually plug in equation (11) into a pointwise regression objective[5], rather than SoftRank, and optimize prototype points and hyperparameters for the sine wave data of Figure 2. The result is shown in Figure 2b, where the prototype points are plotted as blue crosses, and again the predictive mean and two standard deviation functions are shown. We see that a qualitatively similar effect is achieved, using only 10 prototypes, when compared to using a full GP trained using marginal likelihood. The main difference seems to be that a greater linear trend is picked up using this type of training, but the desired growing error bars away from observed data remain.

To implement the SoftNDCG optimization we just need to remember that the π_{ij} (4) are now a function of both the score means and variances:

$$\pi_{ij} \equiv \Pr(S_i - S_j > 0) = \int_0^\infty \mathcal{N}(s|\bar{s}_i - \bar{s}_j, \sigma_i^2 + \sigma_j^2)ds \; . \tag{12}$$

We also need derivatives of $\bar{s}_j$ and σ_j^2 in (11) with respect to prototypes $(\mathbf{X}^p, \mathbf{y}^p)$ and hyperparameters. For this we first need to compute the derivatives with respect to the kernel matrices of (11), and then we need the derivatives of the kernel function (8) with respect to the prototypes and hyperparameters. Splitting up the calculation in this way allows different kernels to be explored with relative ease. Details are omitted due to space limitations, but the computations are straightforward albeit tedious.

5.1 Ranking under uncertainty

One of the major advantages of using a Gaussian process model for the scores is that the individual document uncertainties give us more information to use to decide a ranking at test time. Clearly the vanilla procedure is to simply order according to the predicted score means $\bar{s}_j$. This also corresponds to the most probable ordering under the model described in section 3.1. However, we may decide that we want to be conservative in our predictions, and prefer documents that we are more certain about even if their mean scores are slightly less. Alternatively we may want to be risky and promote documents we are less certain about with the hope that some will turn out to be very good documents. Having uncertainty estimates for the document scores allows us this extra choice. A useful heuristic is to rank according to $\bar{s}_j + \alpha\sigma_j$, where α is negative for a conservative ranking and positive for a risky ranking. Of course, in the spirit of this paper, we actually want to give a ranking that maximizes test NDCG. Since we do not have a proper generative distribution on labels, we cannot compute the optimal ranking. However, loosely speaking, the NDCG metric actually encourages a risky ranking. This is because of the exponential gain function, which rewards top rated documents very highly. All other things being equal, if we are more uncertain about a document there is a bigger chance it might be a top rated document, and therefore might pick up a large reward. We test out some of these ideas in the experiments section.

6 Experiments

We carried out some experiments using the benchmark Learning to Rank (LETOR) OHSUMED data set [13]. The advantages of this data set are that it is publicly available and there are some reference baseline evaluation results for comparison. The LETOR package also provides an evaluation tool in order to ensure that comparative evaluations all use the same calculation.

The OHSUMED data set is a collection of medical publication abstracts. It contains 106 queries, and a total of 16,140 judged query-document pairs, classified as *definitely*, *possibly*, or *not relevant*. 25 standard IR features are extracted from the records. The data comes ready-split into 5 folds each consisting of 60/20/20 splits of the training/validation/test sets. Models are built on each of the five folds using the training set; the validation set can be used to select a model from a set of training runs and/or from a set of models produced within a training run; and the test set is used to evaluate the model.

[5]Pointwise regression objective: $\sum_j \left[\log \sigma_j^2 + \frac{(y_j - \bar{s}_j)^2}{\sigma_j^2}\right]$.

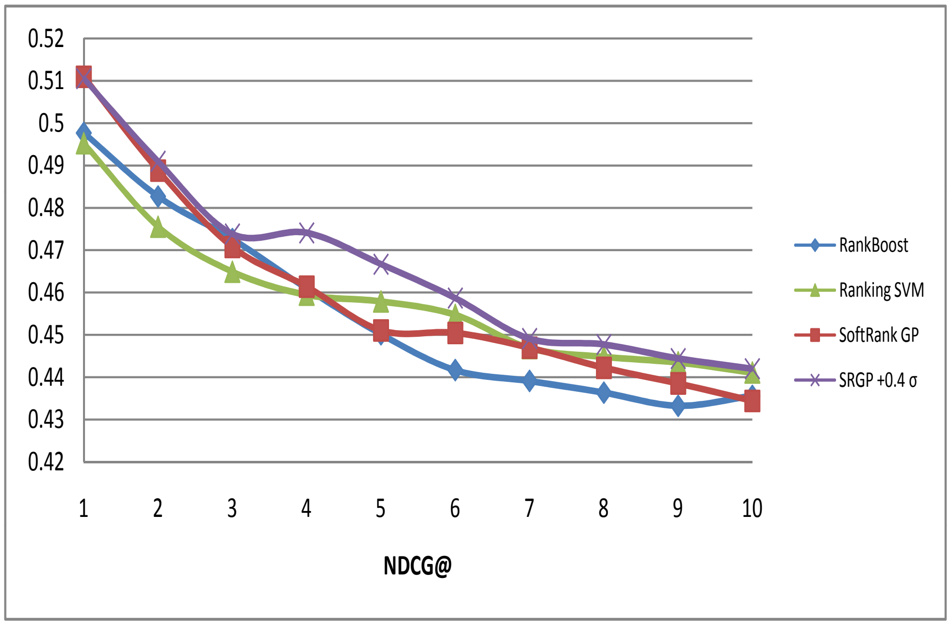


Figure 3: Test NDCG results for the LETOR:OHSUMED data set

We trained the SoftRank GP as described in section 5. There are several design choices that can be made for the model, including the number of prototypes, the type of kernel function, how prototypes are initialised, and how kernel hyperparameters are initialised. Also, as discussed in [8], there is a design choice for the SoftNDCG objective; an argument is made for training with a super-linear discount function, and experiments show that this provides better generalisation to test sets.

In the results shown in Figure 3, 30 prototype points were used, which were initialised from data points randomly chosen except that they were distributed equally among the 3 classes. The virtual target values for these prototypes, which are optimised by the training, were initialised to their observed label values, offset to be zero-mean. An ARD + Linear kernel was used. The length parameters of the ARD sub-kernel were initialised to $0.5\sqrt{Dv_d}$ where v_d is the sample variance of the d^{th} feature. The ARD scale parameter was initialised to the standard deviation of the sample labels. Each linear kernel parameter was initialised to be the inverse of the square of the corresponding ARD length parameter.

A super-linear discount function was used for the SoftNDCG objective. However, the models were selected on the basis of the standard NDCG@10 when run on the validation set for a given fold. An LBFGS algorithm [14] was used to perform the optimisation. Each fold was trained from 5 different initialisations, and the best validated model kept in each case.

Figure 3 shows the test set NDCG for the OHSUMED data set for various truncation levels. The baseline references RankBoost [15] and RankSVM [4, 5] are also plotted. We see that the SoftRank GP gives very competitive performance, and significantly outperforms the reference algorithms at the top of the ranking. This indicates that the SoftRank GP manages to place the highly relevant documents at the top of the ranking. Figure 3 also shows the results for the SoftRank GP where instead of ordering by the predicted mean scores $\bar{s}_j$ alone, we order by $\bar{s}_j + 0.4\sigma_j$, as described in section 5.1. This is a riskier ranking, where we slightly promote uncertain documents. In this case we do obtain higher test NDCG results with the SoftRank GP outperforming the reference algorithms at all truncation levels. However, we need to do a more thorough experimental evaluation to draw any firm conclusions, because we have also seen situations were this simplistic use of variance information detracts from test set performance. The key point is that SoftRank GP provides a framework for using uncertainty information in ranking.

7 Discussion

In this paper we have described a way of combining Gaussian process models with the SoftRank method for directly optimizing NDCG in training. The Gaussian process allows the smoothing uncertainties in SoftRank to reflect modelling uncertainty in the learnt score function. We have shown competitive results on the benchmark LETOR OHSUMED data set. The extra information provided by the uncertainties can be used to provide different styles of rankings, from conservative to risky. This is an interesting direction of research which needs more thorough evaluation.

Although this paper has concentrated on NDCG, the method can be used in conjunction with any metric that can be described in terms of predictive score probabilities. Other discontinuous metrics can be be naturally smoothed in this way, including standard IR metrics such as Average Precision.

A final advantage of the GP approach is that optimizing the ARD lengthscale hyperparameters (8) can allow the model to choose which features are important for the regression and which are irrelevant. For an irrelevant feature the corresponding length-scale hyperparameter will typically grow large, and the feature can be removed if desired. For the web search domain, with potentially huge numbers of features, this could be a useful way to prune unnecessary features and thus speed up computation. We leave this possibility for future work. For the OHSUMED experiments reported here, with only 25 features, the model did not choose to prune any features out.

References

[1] S. Robertson and H. Zaragoza. On rank-based effectiveness measures and optimization. *Information Retrieval*, 10(3):321–339, 2007.

[2] K. Järvelin and J. Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *SIGIR*, 2000.

[3] K. Crammer and Y. Singer. Pranking with ranking. In *NIPS 14*, 2002.

[4] 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.

[5] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of Knowledge Discovery in Databases*, 2002.

[6] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML*, 2005.

[7] C. Burges, R. Ragno, and Q. V. L. Le. Learning to rank with nonsmooth cost functions. In *NIPS*, 2006.

[8] M. I. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: Optimising non-smooth rank metrics. In *WSDM*, 2008. http://research.microsoft.com/˜mitaylor/sigir07LetorSoftRankCam.pdf.

[9] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT press, 2006.

[10] R. M. Neal. Bayesian learning for neural networks. In *Lecture Notes in Statistics 118*. Springer, 1996.

[11] W. Chu and Z. Ghahramani. Gaussian processes for ordinal regression. *Journal of Machine Learning Research*, 6:1019–1041, 2005.

[12] Edward Snelson and Zoubin Ghahramani. Sparse Gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *NIPS 18*, pages 1257–1264. MIT press, Cambridge, MA, 2006.

[13] T-Y Liu. LETOR: Benchmark datasets for learning to rank, 2007. Microsoft Research Asia. http://research.microsoft.com/users/tyliu/LETOR/.

[14] J. Nocedal and S. Wright. *Numerical Optimization, Second Edition*. Springer, 2006.

[15] Y. Freund, R. Iyer, R. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2005.