5-9-2016

Kernelized Cost-Sensitive Listwise Ranking

Gabriela Guimaraes Olinto
ggo5219@rit.edu

R · I · T

Kernelized Cost-Sensitive Listwise Ranking

by

Gabriela Guimaraes Olinto

A Thesis Submitted in Partial Fulfillment of the
Requirements for the degree of Master of Science
in Applied Statistics

School of Mathematical Sciences
College of Science

Rochester Institute of Technology
Rochester, NY
May 9, 2016

Committee Approval

Ernest Fokoue, Associate Professor, School of Mathematical Sciences Thesis Advisor	Date
Joseph Voelkel, Professor, School of Mathematical Sciences Committee Member	Date
Steven LaLonde, Associate Professor, School of Mathematical Sciences Committee Member	Date

"All Knowledge is, in final analysis, History.
All sciences are, in the abstract, Mathematics.
All judgments are, in their rationale, Statistics."
Radhakrishna Rao

Abstract

This thesis research aims to conduct a study on a cost-sensitive listwise approach to learning to rank.

Learning to Rank is an area of application in machine learning, typically supervised, to build ranking models for Information Retrieval systems. The training data consists of lists of items with some partial order specified induced by an ordinal score or a binary judgment (relevant/not relevant). The model purpose is to produce a permutation of the items in this list in a way which is close to the rankings in the training data. This technique has been successfully applied to ranking, and several approaches have been proposed since then, including the listwise approach.

A cost-sensitive version of that is an adaptation of this framework which treats the documents within a list with different probabilities, i.e. attempt to impose weights for the documents with higher cost. We then take this algorithm to the next level by kernelizing the loss and exploring the optimization in different spaces.

Among the different existing likelihood algorithms, we choose ListMLE as primary focus of experimentation, since it has been shown to be the approach with the best empirical performance. The theoretical framework is given along with its mathematical properties.

Experimentation is done on the benchmark LETOR dataset. They contain queries and some characteristics of the retrieved documents and its human judgments on the relevance of the documents on the queries.

Based on that we will show how the Kernel Cost-Sensitive ListMLE performs compared to the baseline Plain Cost-Sensitive ListMLE, ListNet, and RankSVM and show different aspects of the proposed loss function within different families of kernels.

Acknowledgements

It would not have been possible to write this thesis without the help and support of the kind people around me, but only some of whom it is possible to give a particular mention here.

Above all I would like to acknowledge the financial and academic support of the Ministry of Education in Brazil, particularly the Brazil Scientific Mobility Program sponsored by CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) that provide the necessary financial support for the entire degree.

I would like to thank Andre Lobato Ramos for his personal support, love and patience with me this past two years. My parents and brother that have given their unequivocal support, hopes, and love.

A kindly and especial thanks to my academic advisor Professor Ernest Fokoue. This thesis would not be likely to happen without his constant support, knowledge, patience and guidance. His passion, unconditional dedication, knowledge and bright insights made me, every day, a better person and scientist.

I also want to thank my thesis committee Professor Joseph Voelkel and Professor Steven LaLonde for their time, patience, encouragement, and insightful comments.

Last but not least, I would like to thank the entire department staff and chair for their constant assistantship and kindness. All the small talks on the hallways, advises, and shared experiences made RIT and Rochester forever a second home to me. That will not be forgotten, and I can not thank you enough.

Contents

List of Figures

List of Tables

To my beloved family

Chapter 1

Introduction and problem formulation

Ranking is a crucial problem in several different areas, especially in Information Retrieval (IR), Machine Learning and artificial intelligence. Information Retrieval has its foundations on searching automated ways of reducing information overload. Web search engines are one of the most straightforward applications, but universities, government entities, and libraries use these systems to provide easier access to journals, documents, and books.

In a general scenario, the ranking problem can be defined as described by Lan et al. (2009): "Given a group of objects, a ranking model sorts the objects with respect to each other in the group, according to their degrees of relevance, importance, or preferences. Where in IR, a group corresponds to a query, and objects corresponds to documents associated with the query".

In other words, one could imagine a collection of queries with each query containing the n documents to be ranked. These documents could be ranked by a relevance score y where highly relevant documents are more useful and have higher ranks. Thus, one could formulate that the desire in ranking, is to construct a score function $f(x)$ such that the discrepancy (disagreement) between its ranking and the ordering of the relevance is as small as possible.

Of course, "as small as possible" means the crucial need to define a suitable loss function, hopefully, one that is convex for the obvious optimization reasons. This indicates that this problem looks eerily like the tradition regression analysis, but for the traditional regression, the loss function is usually much more manageable and symmetric, whereas with ranking extra care is immediately required.

The form of $f(x)$ then becomes of great interest, just like with traditional regression, and various scenarios may be considered from linear to nonlinear to non parametric to kernelized. Because of that, the nature of the input space can quickly become a problem because the data is usually initially unstructured and is then processed in ways that could lead to mathematical challenges. We will later consider a common practice (Lan et al. 2009) of further preprocessing $|f(x)| < BM$ where BM are normalization constants.

With the recently boom of machine learning many efforts have been created around learning to rank models and its association with IR systems, including a flourishing literature on this fascinating subject of ranking and data competitions like the Yahoo challenge (Chapelle and

Chang 2011) that publicly released two datasets used internally at Yahoo! for learning the ranking function for web search and foster the development of state-of-the-art learning to rank algorithms. Some of the most common ones will be explored in Chapter 2.

These models can be categorized into three main groups: pointwise, pairwise and listwise. Pointwise and pairwise directly transform ranking into ordinal regression, and classification on a single object or object pairs respectively (Li, Wu, and Burges 2007) (Herbrich, Graepel, and Obermayer 1999); and listwise that minimizes the loss function based on the ranked list and the ground truth list (Xia et al. 2008).

This thesis focus on the cost-sensitive listwise approach (Lu et al. 2010) that relies on changing the weighting scheme of the document pairs which are measured by their ranking effectiveness with the Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen 2002). Based on that we propose a theoretical improvement by including a Gramian matrix inside the loss function. The kernel trick allows the implicit computation of feature space calculations with functions (kernels) defined in the input space which will enable high-dimensional operations and consequently greater flexibility (Shawe-Taylor and Cristianini 2004).

The early methods of optimization (minimization) of nonlinear functions were developed from the basic idea of making the algorithm evolve by finding new points located in the direction to which the function decreases compared to the current point.

These optimization methods are compared to each other according to the number of evaluations of the objective function which are required for determining the solution. The fewer necessary assessments, the better the method will be considered. And according to how close the solution provided by this method approaches to the exact solution of the problem. The best method is found by close it gets. Each new point is obtained from one-dimensional optimization process having as a start the previous one.

The first reasonable choice for a search direction in the $k-th$ step, d_k, is the opposite direction to the gradient function in the current point x_k. This choice is justified by the fact that, locally, this is the direction in which the function decreases faster (Snyman 2005), (Kiwiel 2001), (Hiton 2012).

Algorithm 1 Gradient Descent method

1: $x_0 \leftarrow\ rand()$
2: $k \leftarrow 0$
3: **while** not stopping criterion **do**
4: $\quad g_k \leftarrow gradient(f(\cdot), x_k)$
5: $\quad F_k \leftarrow Hessian(f(\cdot), x_k)$
6: $\quad x_{k+1} \leftarrow x_k - F_k^{-1} g_k$
7: $\quad k \leftarrow k + 1$
8: **end while**

The only implicit assumption in the application of this algorithm is that the function $f(x)$ is twice differentiable. Notice that the Hessian indicates the step size to achieve a reduction in f while still making sufficiently fast progress towards the optimal solution. Thus, if the

Hessian is a well-behaved matrix, i.e. is positive definite, it can give optimal step sizes towards the right direction and make the convergence even faster.

On the other hand if it is an ill-conditioned matrix it can do a zig-zag trajectory and take longer to converge to the minimum. In some cases where the Hessian cannot be calculated its value is substituted by the learning rate which is a parameter that determines how fast or slow we will move towards the optimal solution making it treatable but sneak in most cases.

With all that said, Gradient Descent is a perfect tool for optimization Empirical Risk Functionals (Empirical Expected Loss) and will be chosen for optimizing our losses for ranking problems.

Some experiments are conducted to validate this new framework using the dataset MSLR-WEB10K in Microsoft Learning to Rank Datasets inside Letor (Liu et al. 2007). This new framework is called Kernel Cost-Sensitive Listwise MLE (Kernel CS-ListMLE) and we validate it by comparing to its simpler version Cost-Sensitive Listwise MLE (CS-ListMLE) (Lu et al. 2010) and to other two well known techniques in the literature ListNet (Xia et al. 2008) and RankSVM (Joachims 2006). We also explore different aspects of the kernels inside the Kernel CS-ListMLE. All the variants of the CS-ListMLE (plain/kernel) were implemented using the R project (R Core Team 2015), the ListNET can be found on RankLib (Dang, n.d.) on the Lemur project, and RankSVM can be found on SVM^{Rank} (Joachims, n.d.). RankLib and RankSVM are free standalone libraries implemented in Java and Oracle respectively.

This thesis is organized as follows. Chapter 2 illustrates Statistical Learning Machines for Ranking showing its general idea and different proposed techniques. Chapter 3 talks about Listwise Ranking, in particular, various evaluation metrics and details on the implementation. Chapter 4 reports on the new framework, the Kernelized Listwise Ranking. Chapter 5 shows data details, experiments, and comparison results. Chapter 6 outlines future points that could be extended from this thesis and draws conclusions.

Chapter 2

Statistical Learning Machines for Ranking

2.1 Introduction

The process in Information Retrieval (Manning et al. 2008) starts when a user enters a query into the system. This query matches several documents in a collection, maybe with different degrees of relevancy, and these results are ranked. The query can be for example a search string in web search engine and the documents, for example, text documents, images, audio or videos.

Most of this systems compute a numeric score on how well each object in the database matches an individual query and rank the objects according to this value. The top ranking objects are then shown to the user.

The model behind this scoring is what we call Learning to Rank which can be summarized in the Figure 1. This figure was first shown by (Liu Tie-Yan 2008).

In general, users expect the results from a search query to be completed in milliseconds, which makes it very hard to evaluate a ranking model for all documents in a big collection. Thus, first, a small number of potentially relevant documents are identified using simpler retrieval models like the boolean model (Lashkari, Mahdavi, and Ghomi 2009), vector space model (Salton, Wong, and Yang 1975) or BM25 (Robertson et al. 1995) and then a more accurate but computationally more expensive machine-learning model is used to re-rank these documents.

The Boolean model was the first model in the literature of Information Retrieval, and it is still widely used. (Lashkari, Mahdavi, and Ghomi 2009) define a standard Boolean model a model that uses exact matching to match documents to a user input (query) by matching the words in the query using Boolean algebra, where words are combined logically with the boolean operators AND, OR, and NOT.

The vector space model (Salton, Wong, and Yang 1975) is a model that represents text documents/corpora into vector representations of its indexed terms. Having the vector

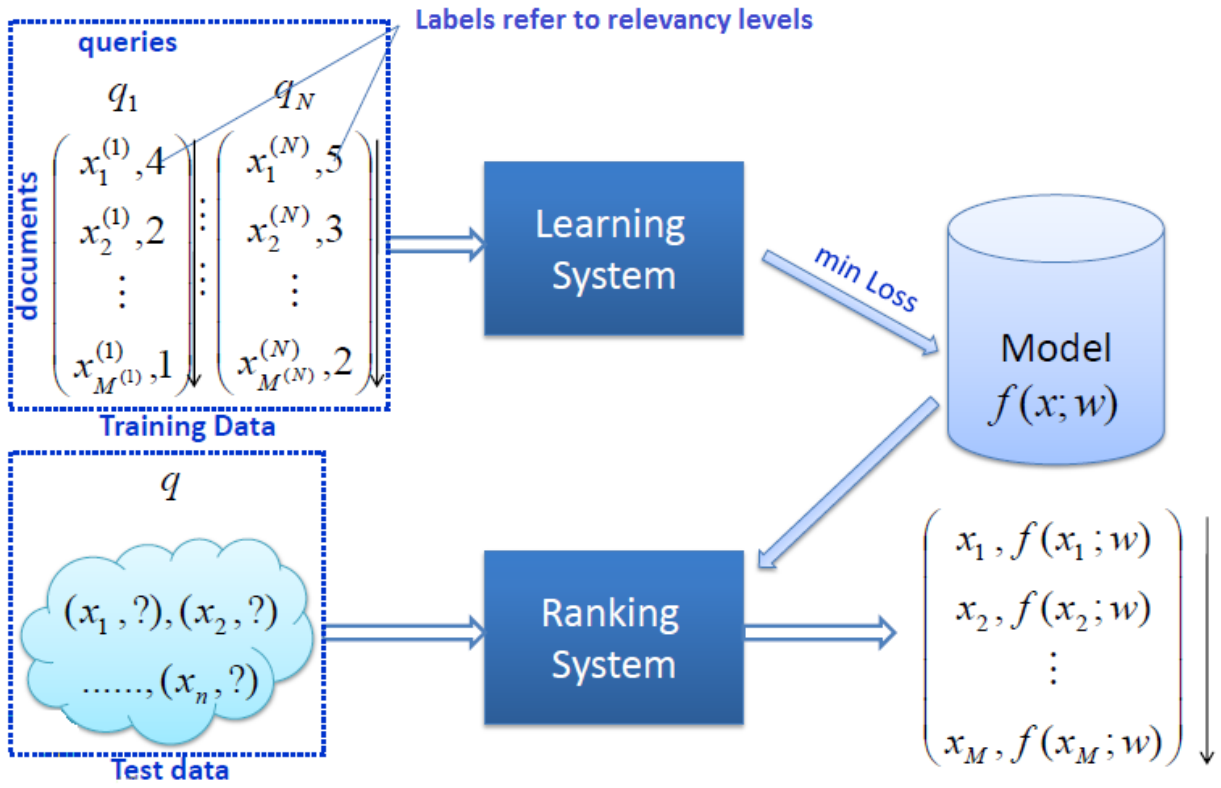


Figure 2.1: Learning to Rank Framework

representation of a document it is easy to apply several mathematical properties to identify how different documents are from each other and from an input query. One famous measure is the cosine similarity (Sidorov et al. 2014), in which the cosine angle between vectors is calculated.

By the time this model was proposed (Salton, Wong, and Yang 1975) the weights for the documents vectors were calculated by term frequency-inverse document frequency (TF-IDF) (Salton and Buckley 1988), (Sparck Jones 1972) score that can be defined as:

$$w_{t,d} = tf_{t,d} \cdot idf_{t,d} = f_{t,d} \cdot \log \frac{\mid D \mid}{1 + \mid \{d \in D : t \in d\} \mid} \tag{2.1}$$

where $tf_{t,d}$ denote the raw frequency of the term t, $\mid D \mid$ is the total number of documents in the corpus and $\mid \{d \in D : t \in d\} \mid$ is the number of documents where the term t appears.

BM25 or also referred as Okapi BM25 (Robertson et al. 1995) is a probabilistic model that estimates a probability of relevance for each query-document pair regardless of semantics. It ranks the documents in relation to a given query in descending order of probability of relevance. Given a query Q, containing keywords $q_1, ..., q_n$, the BM25 score of a document D is:

$$score_{D,Q} = \sum_{i=1}^{n} \log \frac{N - n_{q_i} + 0.5}{n_{q_i} + 0.5} \cdot \frac{f_{q_i,D} \cdot (k_1 + 1)}{f_{q_i,D} + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{avgdl}\right)} \tag{2.2}$$

where $f_{q_i,D}$ is q_i's term frequency in the document D, $\mid D \mid$ is the length of the document D in words, $avgdl$ is the average document length in the text collection from which documents are drawn, k_1 and b are free parameters, usually chosen, as $k_1 \epsilon [1.2, 2.0]$ and $b = 0.75$, N is the total number of documents in the collection and n_{q_i} is the number of documents containing q_i.

Because of the machine-learning models, query-document get feature engineered as numerical vectors or feature vectors that can be divided into three groups:

- Query-independent: depend only on the document but not on the query. For example, features that can be precomputed off-line like document's length or PageRank. PageRank (Page et al. 1999) is an objective measure of citation importance created by link graphs of the web containing as many hyperlinks as possible. It is an algorithm used by Google Search and correspond to people's subjective idea of importance.
- Query-dependent: depend both on the content of the document and the query. For example, TF-IDF in Equation (2.1).
- Query-level: depend only on the query. For example, the number of words in a query.

Some of these features and simpler retrieval models were used at the public available LETOR dataset (Liu et al. 2007).

2.2 Learning to Rank categories

One would say that web search engines are the most popular applications with this framework, and they have been using machine learning models for ranking since the beginning of the century. Google was pioneer using the PageRank idea in 1998. Later Yahoo developed a gradient boosting ranking function in 2003. And after that, Microsoft developed the RankNet (Burges et al. 2005) algorithm for Bing in the earlies of 2005.

This thesis research focus on what was developed by Microsoft Research Asia "Learning to Rank for Information Retrieval"(Liu 2009) and what it has followed after that. Learning to Rank problems can be categorized into three groups by their input representation and loss function:

- Pointwise approach:

In this case, one could assume that each query-document pair in the training data has real/ordinal ranks. Then the learning to rank problem can be approximated by a regression problem, i.e. given a single query-document pair, the score/rank can be predicted. Some existing supervised machine learning algorithms can be used for this purpose like ordinal regression, regression trees, and classification algorithms.

One might say that is only slightly different from conventional machine learning methods, does not directly use the comparison nature of ranking and assumes independent relevance. Some of the unique properties of ranking for Information Retrieval have not been considered either.

- Pairwise approach:

Pairwise ranking aims to predict the relative order between two documents and thus captures the local comparison nature of ranking. It is usually modified to be a binary classification problem where one tries to predict whether one document is preferred over the other. It considers all permutations of documents, and it aims to minimize an average number of inversions in ranking.

- Listwise approach:

Listwise ranking focus on ordering a set (list) of input documents and minimize the inconsistency between the predicted permutations and its ground truth permutation. It captures then the global comparison nature of ranking.

In other words, these algorithms try, over all queries in the training data, to optimize the value of one of the evaluation measures. Since most of the evaluation measures are not continuous functions with respect to ranking model's parameters, it becomes a harder task. Some continuous approximations or bounds on evaluation measures have to been used to solve that problem.

Because ranking is a relatively new problem in machine learning, there are many ranking approaches in the literature that try to reduce the ranking problem to other known learning problems like:

- From pointwise (ordinal) ranking to binary classification:

(Lin and Li 2012) introduced a simple step to convert the binary outputs to a rank, making the generalization straightforward because all the binary classification problems are now solved jointly to obtain a single binary classifier. This showed that low-error binary classifiers can be cast as low-error ordinal rankers.

- From pairwise ranking to binary classification:

Proposed by (Balcan et al. 2008), it transforms the problem with two ordinal outcomes (pairwise ranking) to classification where it learns on whether a is preferred over b and during prediction votes for each example in the test set to rank them.

- From listwise ranking to regression

(Cossock and Zhang 2008) have shown the Subset Ranking algorithm can be viewed as a reduction from listwise ranking to regression, i.e. regression with various cost functions can be used to approximate a Bayes optimal listwise ranker.

Of course, these approximations do not yield as good results and the Learning to Rank algorithms groups. The evolution of this groups has been studied by (Cao et al. 2007) that showed why and how was the transition from pairwise to listwise and (Burges 2010) illustrates the evolution of the listwise algorithms themselves starting with ranknet, going to lambdarank and then to lambdamart.

Some famous examples for pointwise, pairwise and listwise are respectively:

PRank (Crammer, Singer, and others 2001) made a variant of the perceptron algorithm that found multiple parallel hyperplanes separating the various ranks; its output is a weight vector w and a sorted vector of $K-1$ thresholds θ.

$$Y = \begin{cases} 1 & \text{if} \quad y^* \leq \theta_1 \\ 2 & \text{if} \quad \theta_1 < y^* \leq \theta_2 \\ \vdots & \\ K & \text{if} \quad \theta_{K-1} \leq y^* \end{cases}$$

This model tries to predict an output with smallest rank k such that $wx < \theta_k$.

Ranking SVM (Joachims 2006) mapped the similarities between queries and the clicked pages onto a particular feature space, then it calculates the distances between any two of the vectors obtained in the mapping phase and finally forms a similar optimization problem to a standard classification SVM and solves this problem the regular way.

AdaRank (Xu and Li 2007) had the simple idea of constructing "weak rankers" repeatedly based on re-weighted training queries and linearly combining the weak rankers for making ranking predictions. In learning, AdaRank minimizes a loss function directly defined on performance measures.

Chapter 3

Listwise Ranking, metrics and implementation

3.1 Ranking metrics

There are many of standard metrics to judge how well the learn to rank models are doing on training the data and also to compare how they perform between algorithms (Järvelin and Kekäläinen 2002). These metrics take into account the relative order of the documents retrieved by the system and gives more weight to documents returned at higher ranks. Often these models are reformulated as an optimization problem on one of these metrics. Some examples are:

- Mean reciprocal rank (MRR)

The mean reciprocal rank (Voorhees and others 1999) is the average of the reciprocal ranks of results for a sample of queries Q

$$MRR = \frac{1}{\mid Q \mid} \frac{\sum_{i=1}^{|Q|} 1}{rank_i} \tag{3.1}$$

where rank_i refers to the rank position of the first relevant document for the ith query.

- Mean average precision (MAP)

Let's first define Precision and Recall. (Brodersen et al. 2010) defined Precision is the fraction of the documents retrieved that are relevant to a user.

$$Precision = \frac{\mid \{relevant\ documents\} \cap \{retrieved\ documents\} \mid}{\mid \{retrieved\ documents\} \mid} \tag{3.2}$$

And Recall is the fraction of the documents that are relevant to the query that was retrieved.

$$Recall = \frac{\mid \{relevant\ documents\} \cap \{retrieved\ documents\} \mid}{\mid \{relevant\ documents\} \mid} \tag{3.3}$$

Now, consider getting precision as a function of recall $p(r)$, i.e. at every position in the ranked sequence of documents computing a precision and recall. This would allow us to plot a precision-recall curve in which the area would represent the average precision that can be approximated by

$$AvgP = \sum_{k=1}^{n} \frac{p(k)1_{rel(k)}}{\mid \{relevant\ documents\} \mid} \tag{3.4}$$

We can then define the Mean Average Precision, which is the average precision for each query within a set of queries.

$$MAP = \sum_{q=1}^{Q} \frac{AvgP(q)}{\mid Q \mid} \tag{3.5}$$

- Normalized Discounted Cumulative Gain (NDCG)

The Discounted Cumulative Gain (Järvelin and Kekäläinen 2002) measures the gain of a document based on its position in the result list of documents retrieved from a query.

$$DCG = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)} \tag{3.6}$$

Because the length of the query results usually differs from one query to the other, we normalize them by dividing it by the maximum possible DCG on that query so the measure can be comparable between lists. What this means is that we actually need to recalculate the above metric but this time with the best ranked ordered on that list and get:

$$NDCG = \frac{DCG}{max(DCG)} \tag{3.7}$$

3.2 Formalization of Listwise Ranking

Let's give a formalization of the listwise ranking algorithms that will be the main focus in this thesis research. Consider the query-level framework proposed by (Lan, Liu, Qin, et al. 2008):

Let Q, D and X be respectively the query space, the document space and the d-dimensional feature space. Let q be a random variable defined in the query space with unknown

probability distribution P_Q. Let f denote the real value ranking function, which assigns each document a score $f(x)$. The scores of the documents associated with the same query are used to rank the documents. We measure the loss of ranking documents for query q using f with a loss function $L(f;q)$. The goal of ranking is to minimize the expected risk of f

And the extended query idea given by (Lan et al. 2009):

Represent query q by (z, y), where $z = (x_1, \ldots, x_m)$ and y stands for the ground-truth permutation of m documents. Let $Z = X^m$ be the input space, whose elements are m feature vectors corresponding to the m documents, where $y(i)$ stands for the index of the document whose rank is i in the permutation y. We call m the list length and assume that $m \geq 3$. Let Y be the output space, whose elements are permutations of the m documents. Then we regard (z, y) as a random variable on the space $Z \times Y$ according to an unknown probability distribution $P(\cdot, \cdot)$.

The goal of ranking is to minimize the expected risk of f. Thus, rewriting $L(f;q) = l(f;z,y)$ and $P_Q = P(\cdot,\cdot)$, it can be defined as:

$$R_t(f) = \int_{Z \times Y} l(f;z,y)P(dz,dy) \tag{3.8}$$

Since $P(\cdot,\cdot)$ is unknown, the empirical risk is used to approximate the expected risk, and is defined as:

$$\hat{R}_t(f;S) = \frac{1}{n}\sum_{i=1}^{n} l(f;z,y) \tag{3.9}$$

where i denote the i.i.d sampled training data from the space $Z \times Y$,
$S = \{(z_1, y_1), \ldots, (z_n, y_n)\}$ and $z_i = (x_1^{(i)}, \ldots, x_m^{(i)})$.

One of the most critical analyses is the generalization ability is to find a tight upper bound of $sup_{f \epsilon \mathcal{F}}(R_l(f) - \hat{R}_l(f;S))$ which for this case can be obtained by applying the theory of Rademacher Average (Mendelson 2002) that measures how much a function class F can fit random noise and it is showed by the following theorem proofed by (Lan, Liu, Ma, et al. 2008).

Theorem 1. *Let Λ denote a listwise ranking algorithm, and let $l_\Lambda(f;z,y)\epsilon[0,1]$ be its listwise loss, given the training data $S = \{(z_i, y_i), i = 1, \ldots, n\}$, with probability at least $1-\delta$, the following inequality holds:*

$$sup_{f \epsilon \mathcal{F}}(R_l(f) - \hat{R}_l(f;S)) \leq \sqrt{\frac{2ln(2/\delta)}{n}} + 2\hat{\mathcal{R}}(l_\Lambda \circ \mathcal{F}) \tag{3.10}$$

where $\hat{\mathcal{R}}(l_\Lambda \circ \mathcal{F}) = E_\sigma \; sup_{(f \epsilon \mathcal{F})} \; \frac{1}{n}\sum_{i=1}^{n} \sigma_i l_\Lambda(f;z_i,y_i)$

There are three listwise loss functions that are widely use:

$$\textbf{ListMLE} : l(f; z, y) = -logP(y \mid z; f) \tag{3.11}$$
$$P(y \mid z; f) = \prod_{i=1}^{m} \frac{\phi(f(x_{y(i)}))}{\sum_{j=1}^{m} \phi(f(x_{y(j)}))}$$

$$\textbf{ListNet} : l(f; z, y) = -\sum_{\forall \pi \epsilon Y} P(\pi|z; g_y) logP(\pi|z; f) \tag{3.12}$$
$$P(\pi|z; g_y) = \prod_{i=1}^{m} \frac{\phi(g_y(x_{\pi(i)}))}{\sum_{j=1}^{m} \phi(g_y(x_{\pi(j)}))}$$
$$P(\pi|z; f) = \prod_{i=1}^{m} \frac{\phi(f(x_{\pi(i)}))}{\sum_{j=1}^{m} \phi(f(x_{\pi(j)}))}$$

$$\textbf{RankCosine} : l(f; z, y) = \frac{1}{2}\left(1 - \frac{\phi(g_y(z))^T \phi(f(z))}{\|\phi(g_y(z))\| \|\phi(f(z))\|}\right) \tag{3.13}$$

Where $g_Y(x)$ is the score of x given in the ground-truth, $\phi(\cdot)$ is the transformation function, which is an increasing and strictly positive function. Two assumptions are further made on the feature vector and the ranking model for implementation purposes. Let x be the feature vector of a query document-pair, we assume that $\forall x \epsilon X$, $||x|| \leq M$ and the ranking model f to be learned is from the linear function class $F = \{x \to w \cdot x : ||w|| \leq B\}$. Therefore, we have $\forall x \epsilon X$, $\forall f \epsilon F$, $|f(x)| \leq BM$. With this normalization, the algorithms can minimize the empirical risk in learning. Note that with this normalization does not affect their optimal solution.

3.3 Characteristics

(Xia et al. 2008) associates equations (3.11), (3.12) and (3.13), that were developed respectively by (Xia et al. 2008), (Cao et al. 2007), (Qin et al. 2008), with the surrogated losses Likelihood, Cosine and Cross entropy respectively and proposes theoretical analysis of the properties of this surrogate loss functions, including consistency, soundness, continuity, differentiability, convexity and efficiency.

Loss	Consistency	Soundness	Continuity	Differentiability	Convexity	Complexity
Likelihood	√	√	√	√	√	O(n)
Cosine	√	×	√	√	×	O(n)
Cross entropy	√	×	√	√	√	O(n!)

Table 3.1: Comparison between surrogate losses

The results are summarized on the Table 3.1. As we can see the Likelihood surrogate loss or ListMLE is the one with the best properties and it will be the of main focus study in this research. Because of that, just the properties for ListMLE showed by (Xia et al. 2008) will be presented.

Theorem 2. *Let $l(f; z, y)$ be an order sensitive loss function on $\omega \subset R^n$. $\forall n$ objects, if its permutation probability space is order preserving with respect to $n-1$ objective pairs $(j_1, j_2), (j_2, j_3), \ldots, (j_{n-1}, j_n)$. Then the loss $l(f; z, y)$ is consistent with respect to* (3.8).

Consistency is about whether the obtained ranking function can converge to the optimal one through the minimization of the empirical loss when the training sample size goes to infinity. For the ListMLE case, its loss is order sensitive, therefore, according to Theorem 2, it is consistent.

Soundness is about whether the loss function can indeed represent the loss in ranking, i.e. an incorrect ranking should receive a larger penalty than a correct ranking, and that should reflect the confidence of the ranking.

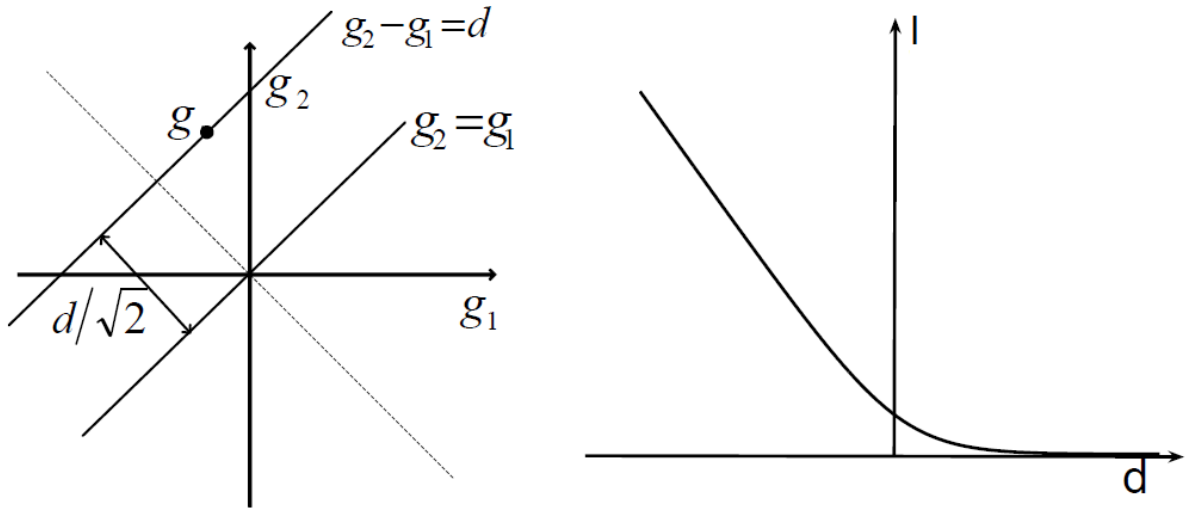


Figure 3.1: Left ranking scores of predicted result, right loss l v.s. d for the likelihood loss.

Again, considering the ListMLE case and looking at Figure 2, suppose that there are two objects to be ranked and they receive scores of g_1 and g_2 from a ranking function. Suppose

that the first object is ranked below the second object in the ground truth. Then the upper left area above line $g_2 = g_1$ on the left graph corresponds to correct ranking and the lower right area to incorrect ranking. According to the definition of likelihood loss, all the points on the line $g_2 = g_1 + d$ has the same loss. Therefore, one can say that the likelihood loss only depends on d. The right graph shows this relationship: loss function l decreases monotonously as d increases which indicate that it penalizes negative values of d more heavily than positive ones. This will make the learning algorithm focus more on avoiding incorrect rankings. Thus, the likelihood loss or more specific the ListMLE function is sound.

The other properties, continuous, differentiable and convex can be verified by (Boyd and Vandenberghe 2004). The likelihood loss can be computed efficiently, with time complexity of linear order to the number of objects.

Some possibilities for ϕ are:

Linear: $\phi_L(x) = ax + b$

Sigmoid: $\phi_S(x) = \frac{1}{1+\exp -ax}$

Exponential: $\phi_E(x) = \exp ax$

Probit: $\phi_P(x) = \Phi(x) = \int_{-\infty}^{x} \frac{\exp -x^2/2}{\sqrt{2\pi}} dx = \frac{1}{2}\left[1 + erf\left(\frac{x}{\sqrt{2}}\right)\right]$

3.4 Cost-Sensitive Listwise Ranking

As explained in the section before, the listwise approach minimizes the loss function defined between the ranked and the ground truth list. This scenario treats all the documents as if they had equal probability, however, the documents with higher ranks should be emphasized. Thus, a cost-sensitive listwise approach was defined by (Lu et al. 2010). This framework redefines the probability by imposing weights for the documents pairs. These weights are computed based on Normalized Discounted Cumulative Gain (NDCG).

The Normalized Discount Cumulative Gain evaluates the performance of the top documents in the ranked list. Suppose n candidate documents are retrieved for a query and each candidate document d_i is represented as a pair (x_i, y_i), where x_i denotes the query-document pair feature vector and y_i is the relevance level. The NDCG@k can be defined as:

$$NDCG@k = \frac{DCG_\pi@k}{DCG_g@k} = \frac{\sum_{j=1}^{n} \frac{2^{y_j}-1}{\log_2(1+\pi(j))} I[\pi(j) \leq k]}{\sum_{j=1}^{n} \frac{2^{y_j}-1}{\log_2(1+g(j))} I[g(j) \leq k]} \tag{3.14}$$

where k denotes the truncation level, π is the ranked list generated by a ranking function and g denotes the ground truth list obtained in the document relevance level descending order. $g(j)$ and $\pi(j)$ denote the ground truth ranking position and the ranked position of the document d_j respectively. $I[x]$ is an indicator function and yields one if x is true and zero otherwise. Assume that the ranking function is f, then the ranked position $\pi(j)$ is computed from:

$$\pi(j) = 1 + \sum_{i=1}^{n} I[f(x_i) > f(x_j)] \tag{3.15}$$

where $f(x_i)$ denotes the rank score of the document d_i. If there are many documents sharing the same relevance level with the document d_j, it is impossible to state the definite value of $g(j)$. So the approximate value $\hat{g}(j)$ is given as:

$$\hat{g}(j) = 1 + \sum_{i=1}^{n} I[y_i > y_j] \tag{3.16}$$

Because of that approximation we can say that $\hat{g}(j) \leq g(j)$ for each j, which implies $\mathrm{DCG}_g@\mathrm{k} \leq DCG_{\hat{g}}@k$. Thus the NDCG@k loss can be written as:

$$L_{ndcg@k} = 1 - NDCG@k \leq \frac{1}{DCG_{\hat{g}}@k}(DCG_{\hat{g}}@k - DCG_{\pi}@k) \tag{3.17}$$

(Lu et al. 2010) showed that the cost-sensitive listwise losses are the upper bound of NDCG loss. For notation simplicity, let $a(i) = 2^{y_i} - 1$, $b(j)$ and its gradient $\Delta b(j)$ being:

$$b(j) = \begin{cases} 1/\log_2(1+j) & j \leq k \\ 1/\log_2(1+k) & j > k \end{cases}$$

$$\Delta b(j) = \begin{cases} \frac{-\log 2}{(1+j)[\log(1+j)]^2} & j \leq k \\ 0 & j > k \end{cases}$$

We can rewrite NDCG@k loss as:

$$L_{ndcg@k} \leq \frac{1}{DCG_{\hat{g}}@k}\left(\sum_{i=1}^{n} a(i)(b(\hat{g}(i)) - b(\pi(i)))\right) + \frac{1}{DCG_{\hat{g}}@k}\sum_{i=1}^{n}\frac{a(i)}{\log_2(1+k)}$$

$$C_1 = \frac{1}{DCG_{\hat{g}}@k}\sum_{i=1}^{n}\frac{a(i)}{\log_2(1+k)} \quad C_2 = C_1 + \frac{1}{DCG_{\hat{g}}@k}\sum_{y_j=y_i}[-a(j)\Delta b(\hat{g}(j)) - a(i)\Delta b(\hat{g}(i))]$$

$\therefore$ b is a convex function $\iff b(x) - b(y) \leq \Delta b(x)(x - y)$

$$L_{ndcg@k} \leq \frac{1}{DCG_{\hat{g}}@k}\sum_{j=1}^{n} a(j)\Delta b(\hat{g}(j))(\hat{g}(j) - \pi(j)) + C_1$$

$$= \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} a(j)(-\Delta b(\hat{g}(j)))\{\left(1 + \sum_{i=1}^{n} I[f(x_i) > f(x_j)]\right) - \left(1 + \sum_{i=1}^{n} I[y_i > y_j]\right)\} + C_1$$

$$\therefore I[x > 0] - I[y > 0] \leq I[xy < 0] + I[y = 0]$$

$$L_{ndcg@k} \leq \frac{1}{DCG_{\hat{g}}@k} \sum_{y_j > y_i} [-a(j)\Delta b(\hat{g}(j)) - a(i)\Delta b(\hat{g}(i))] I[f(x_j) < f(x_i)] + C_2 \tag{3.18}$$

This shows that the weights of the document pairs of each query can be calculated at once in training and justifies the correlation between cost-sensitive listwise loss and NDCG@k loss. Considering that z_j and w_j are non negative and

$$I[z > 0] \leq \log_2(1 + \exp(z))$$

$$\sum_{j=1}^{n} w_j \log_2(z_j) \leq \left(\sum_{j=1}^{n} w_j\right) \cdot \left(\log_2 \left(\sum_{j=1}^{n} w_j z_j / \sum_{t=1}^{n} w_t\right)\right)$$

we can get the following theorem.

Theorem 3. *The cost-sensitive listwise loss is the upper bound of NDCG loss* $L_{ndcg@k}$

$$L_{ndcg@k} \leq \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \beta_j \log_2 \left(\sum_{j=1}^{n} \alpha_{i,j}/\alpha_{i,i} \exp\left(\phi(f(x_j)) - \phi(f(x_i))\right)\right) + C_2$$

where $\alpha_{i,j}$ is the weight of the document d_j and $\alpha_{i,j}/\alpha_{i,i}$ is the weight of the document pair (d_i, d_j). The theorem states definite values for all the weights β_j and α_j.

From that, (Lu et al. 2010) point out a case called Cost-Sensitive ListMLE Ranking Approach for which the loss function on a query can be defined as:

$$L = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \beta_j \log_2 \left(1 + \sum_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp(f(x_t) - f(x_j))\right) \tag{3.19}$$

where the ranking function (score) is linear, i.e. $f(x)$ is the inner product between x and w (model parameters), i.e. $f(x) =< w, x >= w^T x$. Based on that a cost-sensitive approach named Kernelized Cost-Sensitive ListMLE is proposed. To optimize this loss function we need to do gradient descent which is a method that works optimally when is based on the

gradient and the hessian. Let $G(w)$ be the gradient then with the loss function L, $G(w) = (G(w_1), \ldots, G(w_p)))$ can be written as:

$$G(w_l) = \frac{\partial L}{\partial w_l} = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \frac{\beta_j}{log(2)} \frac{\sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[(x_t - x_j)w_l](x_t - x_j)}{\left(1 + \sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[(x_t - x_j)w_l]\right)} \tag{3.20}$$

and let $H(w)$ be the hessian then with the loss function L, $H(w) = (H(w_1), \ldots, H(w_p)))$ can be written as:

$$H(w_l) = \frac{\partial^2 L}{\partial w_l^2} = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \frac{\beta_j}{log(2)} \frac{\sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[(x_t - x_j)w_l](x_t - x_j)^2}{\left(1 + \sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[(x_t - x_j)w_l]\right)^2} \tag{3.21}$$

where l is the l^{th} model parameter. Since it is convex, the model parameters converge to a global optimal solution.

The Algorithm for Gradient Descent in CS-ListMLE is:

Algorithm 2 GD for CS-ListMLE

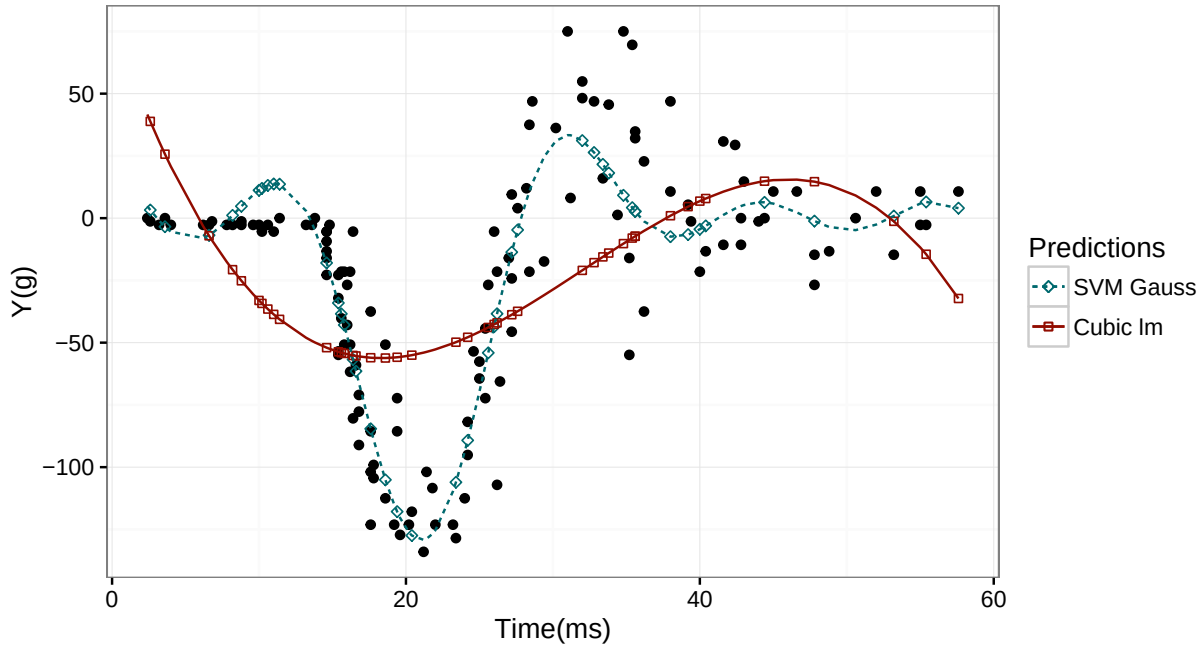
1: Input normalized training set, tolerance rate ϵ, NDCG@k
2: Initialize w and compute weights with respect to each query, normalizing w each step
3: Do Gradient Descent until the change of the loss function is less than ϵ
4: Return w

Chapter 4

Kernelized Listwise Ranking

Although the present Listwise MLE appears to work very well, its use of a linear representation $f(x) =< w, x >= w^T x$ for the score function could be a limitation. Kernels can help capture an arbitrarily nonlinear function by mapping into a higher dimension space.

Let's start considering a regression problem with explanatory variables taken from the motorcycle data introduced with the Matlab XTAL package (Franc and Hlavác 2004). We illustrate the use of SVM regression
(Cortes and Vapnik 1995) with a Gaussian kernel and a Cubic Linear Regression (Stigler 1974).



There are many algorithms that can try to solve these tasks, but clearly any linear machine will fail the task of doing a good job at fitting the response. This is the initial motivation for the so-called kernel trick problem, i.e. any linear model can become a non-linear model by replacing its features by a transformation over pairs of data.

Our first intuition is to do so by a feature mapping, i.e. transform each feature vector by a

user-specified feature map. The kernel trick allows us to do that in a more efficient way; instead of a transformation for each predictor, we apply a single similarity function over the data and enable them to operate in a higher dimensional space without ever computing a transformation for each of the coordinates. Instead we can simply compute the inner product between the images of all pairs of data in the feature space.

It is important to notice that in Figure 3, the SVM fit does a much better job compared to the cubic linear regression one. But the SVM could be even get better. Tunning parameters and different families of kernels can be considered for that end.

Mathematically, let the vector $x \epsilon R^p$ implicitly be projected (mapped) onto a feature space F (of potentially infinite dimension) via a transforming function or mapping η such that $x \to \eta(x)$, and $K(x_i, x_j) = \eta(x_i)^T \eta(x_j)$ (Shawe-Taylor and Cristianini 2004). Many common choices of kernels work very well in practice, some frequently used ones are:

- **Polynomial**: $K(x_i, x_j) = (b x_i^T x_j + a)^d$
- **Gaussian**: $K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
- **Laplace**: $K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right)$
- **Hyperbolic Tangent**: $K(x_i, x_j) = \tanh(b x_i^T x_j + a)$

With all that said it is easy to adapt the Cost-Sensitive Ranking for a Kernel Cost-Sensitive Ranking by kernelizing the ranking function $f(x) =< \theta, x >= \theta^T \eta(x)$, i.e., let

$$\begin{cases} \eta: & X \to F \\ & x \to \eta(x) \end{cases}$$

so $f(x) = \theta^T \eta(x)$ in F. Then $f(x) = \sum_{j=1}^n \theta_j K(x, x_j)$ where $K(\cdot, \cdot)$ is the kernel trick and $K(x_t, x_l) =< \eta(x_t), \eta(x_l) >$ which make it inherit all its good characteristics. The loss function on a query can be defined as:

$$L = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \beta_j \log_2 \left(1 + \sum_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp(\sum_{l=1}^{n} \theta_l K(x_t, x_l) - \sum_{l=1}^{n} \theta_l K(x_j, x_l))\right) \quad (4.1)$$

The Gradient Descent method can be used to optimize the loss function which is a method that works optimally when is based on the gradient and the hessian. It is vital to assess the properties of the Hessian matrix because it's positive definiteness is a blessing on estimating the vector *theta*.

Thus, let $G(\theta)$ be the gradient then with our loss function L, $G(\theta) = (G(\theta_1), \ldots, G(\theta_p)))$ can be written as:

$$G(\theta_l) = \frac{\partial L}{\partial \theta_l} = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \frac{\beta_j}{log(2)} \frac{\sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)] \sum\limits_{l=1}^{n} [K(x_t, x_l) - K(x_j, x_l)]}{\left(1 + \sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)]\right)} \tag{4.2}$$

and let $H(\theta)$ be the hessian then with the loss function L, $H(\theta) = (H(\theta_1), \ldots, H(\theta_p)))$ can be written as:

$$H(\theta_l) = \frac{\partial^2 L}{\partial \theta_l^2} = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \frac{\beta_j}{log(2)} \frac{\sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)] \left(\sum\limits_{l=1}^{n} [K(x_t, x_l) - K(x_j, x_l)]\right)^2}{\left(1 + \sum\limits_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)]\right)^2} \tag{4.3}$$

Considering

$$f(\cdot) = \sum_{l=1}^{n} \theta_l \eta(\cdot) \eta^T(x_l) = \sum_{l=1}^{n} \theta_l K(\cdot, x_l) \tag{4.4}$$

The Algorithm for Gradient Descent in Kernelized CS-ListMLE is:

Algorithm 3 GD for Kernelized CS-ListMLE

1: Input normalized training set, tolerance rate ϵ, NDCG@k
2: Initialize θ and compute weights with respect to each query, normalizing θ each step
3: Do Gradient Descent until the change of the loss function is less than ϵ
4: Return θ

For the kernels used in this case in which the results are shown in Chapter 5, the Hessian appears to behave very well, making the convergence faster, especially in concert with the normalization of the input space. Since this version is based on the same surrogate loss is safe to say that the theoretical properties of the listwise carry over to the kernelized version.

It is important to point out that this nonlinear approach benefits from higher dimensional projections which makes an easy task to fit a broad range of functions since it can search the space better. Although many scientific processes can be described well using linear models, others are inherently nonlinear. Another benefit is efficient use of data. Nonlinear regression like models can produce good estimates of the unknown parameters in the model with relatively small data sets.

The major cost of moving to nonlinear from simpler modeling techniques is the need to use iterative optimization procedures to compute the parameter estimates. The use of iterative procedures requires the user to provide starting values for the unknown parameters before the software can begin the optimization and it is directly tied up to its convergence. Bad starting values can also cause the software to converge to a local minimum rather than the global minimum that defines the least squares estimates.

In our case, there is no analytically solution for the Listwise approaches and the convexity of the loss working in tandem with normalization of the space make this approach very appealing.

Chapter 5

Data Description and Processing

To prove our results we use one of the datasets in the LETOR project. The LETOR dataset is a benchmark collection for research on learning to rank for information retrieval, released by Microsoft Research Asia (Qin et al. 2010). By doing this, the group facilitates the research on learning to rank since no selected queries for training and test, no standard feature vectors, no baseline algorithms and no standard evaluation tools were defined before, making the comparison among different methods not possible.

LETOR was constructed based on multiple data corpora and query sets. The documents were sampled and features were extracted for each query-document pair. The data was partitioned into five folds envisioning cross validation and standard evaluation measures were provided. In addition, some baseline results were provided, to serve as comparison for newly developed methods.

The first version of LETOR was released in April 2007 and used in the SIGIR 2007 workshop on learning to rank for information retrieval. Based on the feedback gather from the IR community, other versions of LETOR became available to the public in 2008 and 2009.

5.1 Cross validation

Fold	Training set	Validation set	Test set
1	$\{S_1, S_2, S_3\}$	S_4	S_5
2	$\{S_2, S_3, S_4\}$	S_5	S_1
3	$\{S_3, S_4, S_5\}$	S_1	S_2
4	$\{S_4, S_5, S_1\}$	S_2	S_3
5	$\{S_5, S_1, S_2\}$	S_3	S_4

Table 5.1: Partitioning for five-fold Cross Validation

(Qin et al. 2010) partitioned each dataset into five parts with about the same number of queries, denoted as S_1, S_2, S_3, S_4 and S_5, for five-fold cross validation. Each fold, had three parts for training, one part for validation, and the last part for test as shown in Table 5.1.

They describe the training set to be used to learn ranking models, the validation set to tune the hyperparameters and the test set to evaluate the performance of the models.

5.2 Baseline models and measures

They tested linear regression as a baseline example for Pointwise approach, RankBoost for Pairwise approach, ListNet for Listwise and measure the quality of these algorithms by Precision (3.2), MAP (3.5), Normalized Discount Cumulative Gain (3.7) measures.

5.3 Features

Their most recent release was the $MSLR - WEB10K$ with 10000 queries and 136 features for each query-url pair which are available to download (Qin et al., n.d.).

Taking a closer look in the data files, each row corresponds to a query-url pair. The first column is relevance label of the pair which goes from 0 (irrelevant) to 4 (perfectly relevant), the second column is query id, and the following columns are features.

The features include covered query term number and ratio, stream length, IDF (Inverse document frequency), sum/min/max/mean/variance of term frequency, sum/min/max/mean/variance of stream length normalized term frequency, sum/min/max/mean/variance of tf∗IDF, boolean model, vector space model, BM25, LMIR.ABS (Language model approach for information retrieval with absolute discounting smoothing), LMIR.DIR (Language model approach for IR with Bayesian smoothing using Dirichlet priors) and LMIR.JM (Language model approach for IR with Jelinek-Mercer smoothing), all by body, anchor, title, url, whole document. Plus number of slash in URL, length of URL, inlink number, outlink number, PageRank, SiteRank (Site level PageRank), QualityScore (The quality score of a web page. The score is outputted by a web page quality classifier.), QualityScore2 (The quality score of a web page. The score is outputted by a web page quality classifier, which measures the badness of a web page.), query-url click count (The click count of a query-url pair at a search engine in a period), url click count (The click count of a url aggregated from user browsing data in a period) and url dwell time (The average dwell time of a url aggregated from user browsing data in a period).

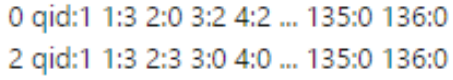


Figure 5.1: Example of two rows from MSLR-WEB10K dataset

5.4 Experiments and Results

In this section we analyze the performance of ListNet, RankSVM, Plain Cost-Sensitive MLE and Cost-Sensitive MLE Kernel polynomial, gaussian, laplace and hyperbolical tangent on the LETOR dataset $MSLR - WEB10K$. This data set contains 5 folds so the following results are averages of this results.

We also explore some different parametrizarion for the kernels, one of them we call **special** in which the offset is considered 0, the polynomial degree is 3 and the bandwidth is $1/p = 1/136 = 0.0074$ inspired by (Meyer et al. 2015). In the other cases we use the vanilla parametrization with offset is considered 0, the polynomial degree is 1 and the bandwidth is 1.

It is important to notice when generating test results that ListNet and Plain Cost-Sensitive MLE optimize distances while RankSVM and Cost-Sensitive MLE Kernel optimize similarities which means different interpretations for the coefficients. In the experiments, the truncation level k takes 3, 5 and 10 as in (Lu et al. 2010).

nDCG	ListNet	RankSVM
@3	**0.6028**	0.4662
@5	**0.6243**	0.5120
@10	**0.6116**	0.5325

Table 5.2: Test results for nDCG@3, nDCG@5 and nDCG@10 on ListNet and RankSVM

Methods	at.3	at.5	at.10	special
Gaussian	0.5763	0.5736	0.5693	0.5548
Laplace	0.5765	0.5673	0.5705	0.5637
Polynomial	**0.6011**	**0.6040**	0.6178	0.5126
Tanh	0.5824	0.5904	0.5666	**0.5743**
Plain	0.5817	0.5984	**0.6316**	

Table 5.3: Test results for nDCG@3 on Kernel and Plain Cost Sensitive List MLE

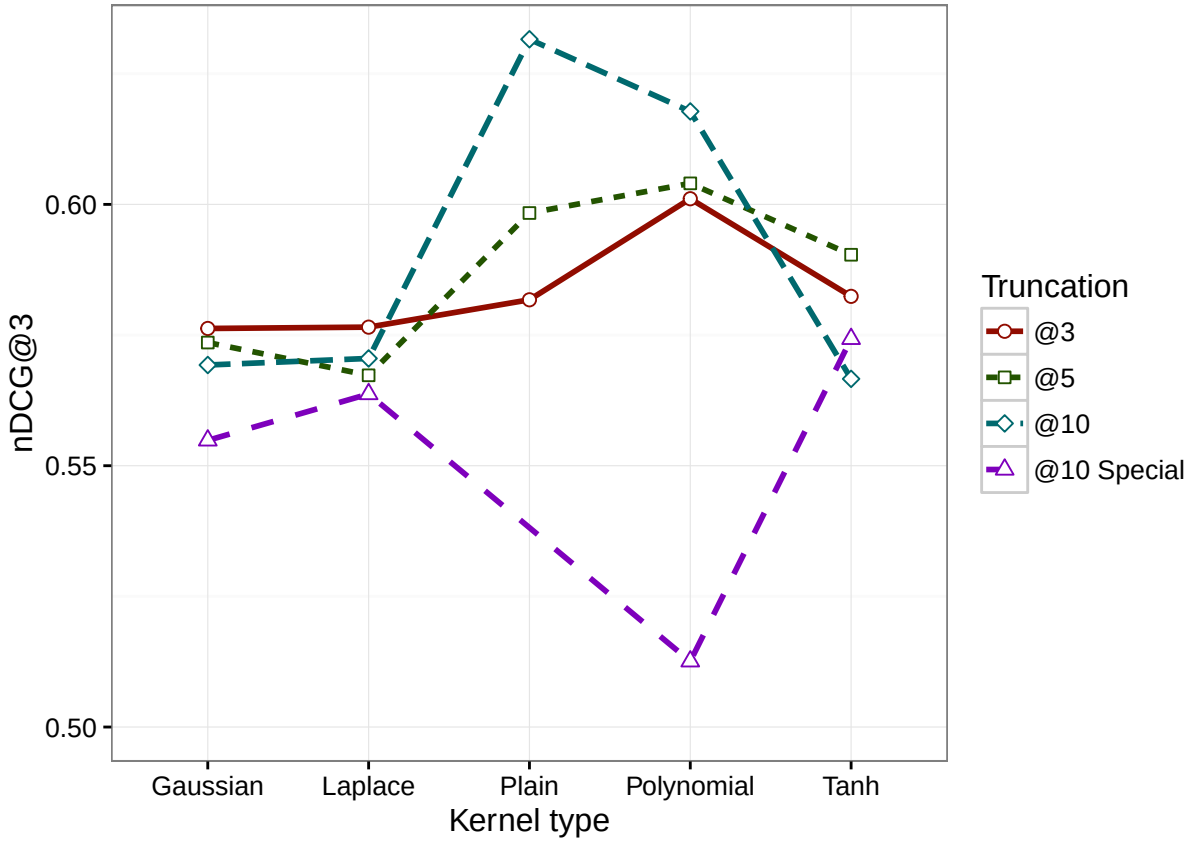


Figure 5.2: Summary for nDCG@3 on Kernel and Plain Cost Sensitive List MLE

Methods	at.3	at.5	at.10	special
Gaussian		0.5948	0.5933	0.5755
Laplace		0.5882	0.5984	0.5850
Polynomial		**0.6177**	0.6293	0.5455
Tanh		0.6058	0.5947	**0.5986**
Plain		0.6122	**0.6425**	

Table 5.4: Test results for nDCG@5 on Kernel and Plain Cost Sensitive List MLE

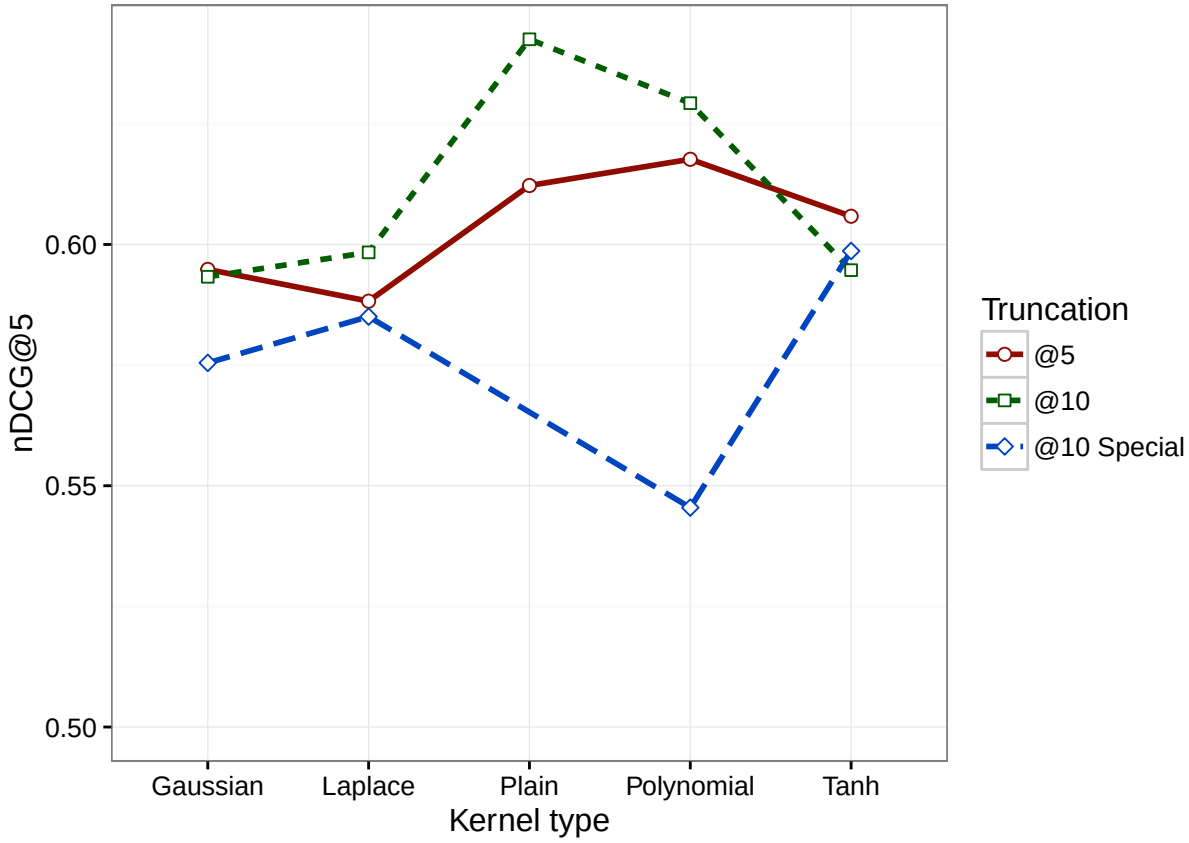


Figure 5.3: Summary for nDCG@5 on Kernel and Plain Cost Sensitive List MLE

Methods	at.3	at.5	at.10	special
Gaussian			0.5860	0.5757
Laplace			0.5876	0.5813
Polynomial			0.6175	0.5579
Tanh			0.5941	**0.5917**
Plain			**0.6276**	

Table 5.5: Test results for nDCG@10 on Kernel and Plain Cost Sensitive List MLE

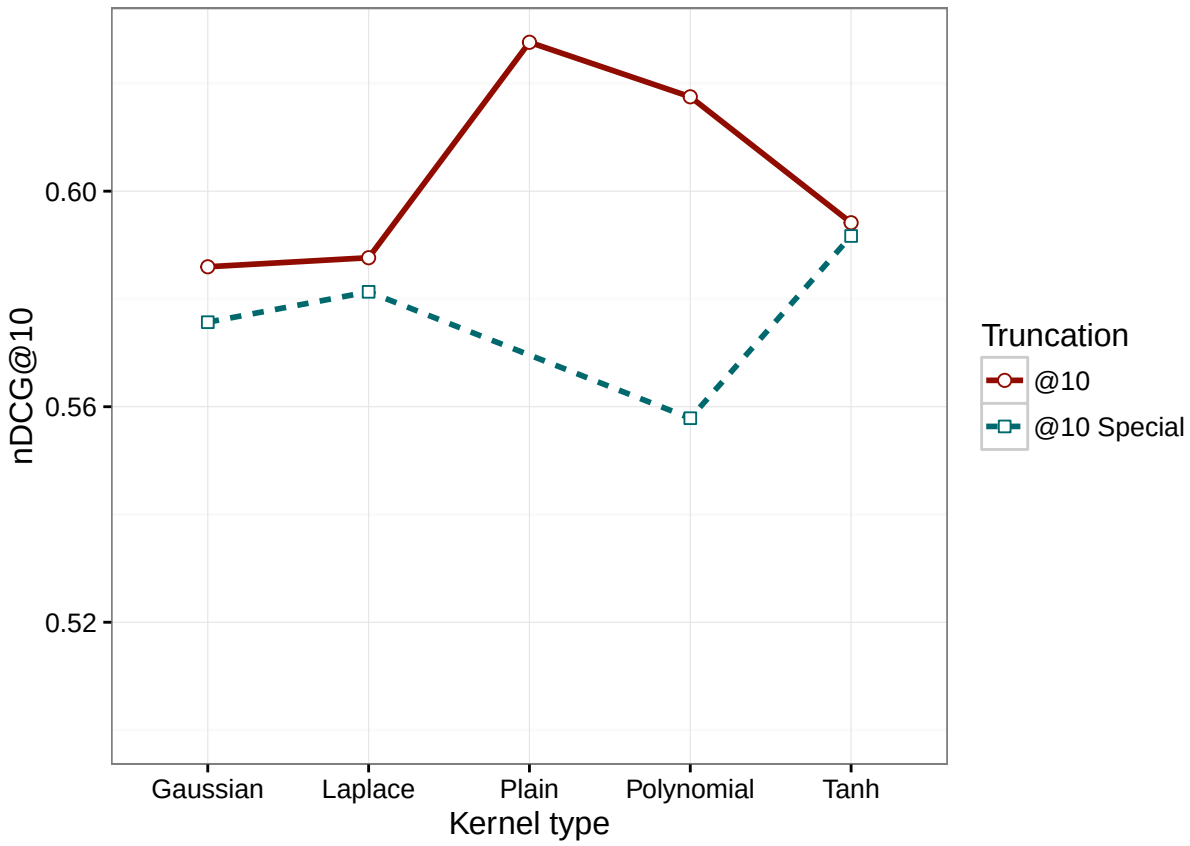


Figure 5.4: Summary for nDCG@10 on Kernel and Plain Cost Sensitive List MLE

Methods	at.3	at.5	at.10	special
Gaussian	7	**3**	4	3
Laplace	**2**	75	**2**	3
Polynomial	3	5	6	**2**
Tanh	11	20	**2**	17
Plain	**2**	**3**	4	

Table 5.6: Iterations until convergence on Kernel and Plain Cost Sensitive List MLE

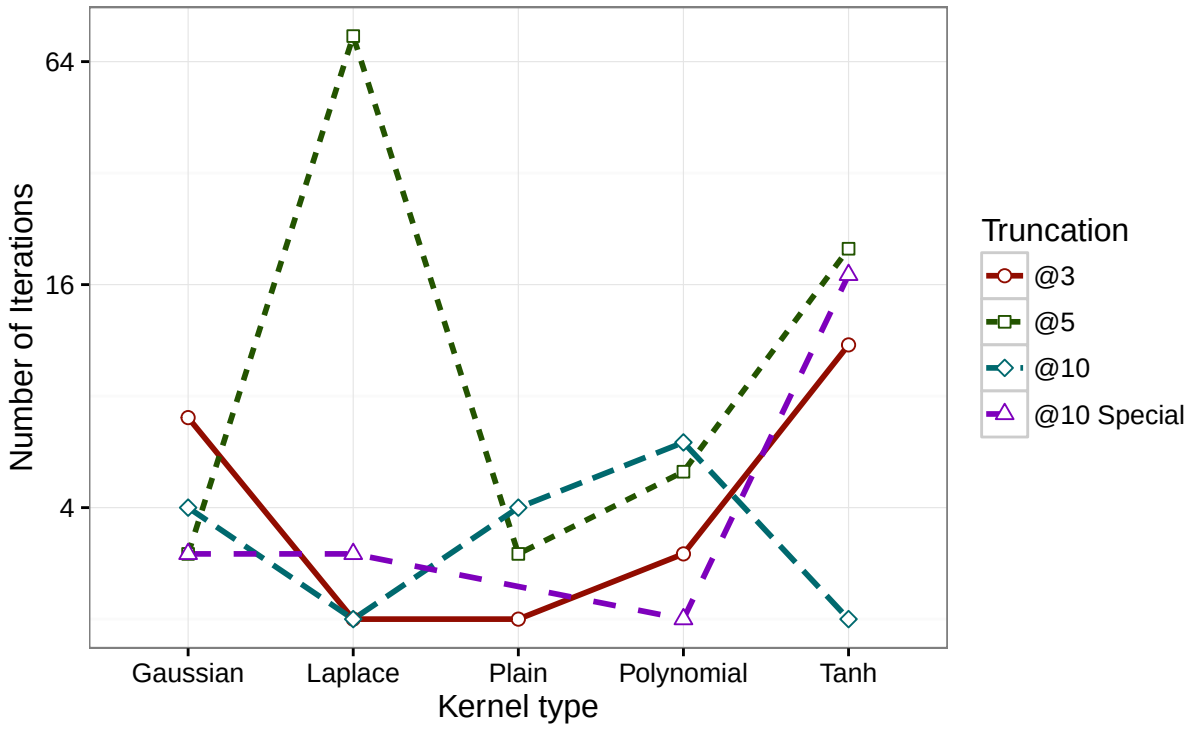


Figure 5.5: Iterations until convergence on Kernel and Plain Cost Sensitive List MLE

Note that the empty values on the tables are because the evaluation metric can just evaluate cases where its truncation value is equal or bigger than training truncation level. At the special column, no parameter tunning can be done for the Plain version, so it is always empty.

Since ListNet and RankSVM do not truncate during training we should compare our results with highest truncation at training (at 10) and same nDCG metric to have somewhat fair comparations. With that said one could notice that CS-ListMLE plain and CS-ListMLE polynomial always perform better then them which reforce the point of our study.

Looking at the test results for nDCG@3, we can see for a truncation level during training also at 3, that the cost-sensitive listwise Polynomial outperforms the others. Considering higher truncation levels but same metric, Polynomial, Plain and tanh outperforms the others at 5, 10 and 10 special respectively.

For test results for nDCG@5 with the same truncation level 5, the Polynomial kernel again outperform the others. Considering higher truncation at training as 10 and special, Plain and Tanh outperforms the others respectively.

Finally, the test results for nDCG@10 shows that the Plain version outperforms with truncation level at 10 although Polynomial kernel also does a very good job and for the special case, Tanh as usual does the best job.

Taking a look at how many iteration it takes to converge, we can say that it is an algorithm that converge relatively fast which is probably due to the fact that the loss function is convex and the Gradient Descent include the second derivative. Considering the truncation level 3, Laplace and Plain are the fastest, at level 5 Gaussian and Plain, at 10 Laplace and Tanh and at 10 special we have got Polynomial.

This results take us to conclude that CS-MLE Plain is more robust when evaluating larger truncation levels than it was trained on but the CS-MLE Kernel outperforms cases in which the truncation and the evaluation are the same, that means, it learns better its own space.

More parameters could have been tunned but since our goal was to show the potential behind the Kernelized options we can see how much could be gain by running way from its vanillas version. Specially for the Tanh kernel, its tests results seem to gain the most.

Chapter 6

Conclusion and Future Work

This thesis studied new approaches for cost-sensitive listwise to learning to rank which is an area of application in machine learning that built ranking models for Information Retrieval systems. The model purpose is to produce a permutation of items that have some partial order induced by an ordinal score.

The listwise approach learns a ranking function by taking lists and minimizing a loss function based on the ranked list and the ground truth list (supervised annotated judgment). These losses are classified in different groups which different inherent properties: we verify that the surrogate likelihood loss is the best one and study the particular ListMLE case.

A cost-sensitive version is the next straightforward step, using the traditional nDCG@k metric, which we generalize by introducing the Gram matrix to the loss or kernelizing it. The theoretical framework was given along with their mathematical properties. Four families of kernels were considered: Gaussian, Laplace, Polynomial and Hyperbolic Tangent. We use their vanilla version to compare with the baseline plain method as well as some different parametrizarion common utilized for the kernels in which we call "special". The carry parameters set as offset 0, polynomial degree 3 and bandwidth $1/p = 0.0074$.

The experimentation is done on the benchmark LETOR $MSLR - WEB10K$ dataset. It is the most recent available dataset released by the Microsoft group. They contain queries and some characteristics of the retrieved documents and its human judgments on the relevance of the documents with respect to the queries and its natural five fold set allowed cross validation.

Gathering all those ideas we have shown some higher performance Kernel Cost-Sensitive ListMLE compared to the baseline ListMLE and compared different aspects of the proposed loss function within different families of kernels.

The results show that CS-MLE Plain is more robust when evaluating larger truncation levels than it was trained on, but the CS-MLE Kernel outperforms cases in which the truncation and the evaluation are the same, that means, it learns better its space.

More grid values for parameters could have been studied but since our goal was to show the potential behind the Kernelized options we could show how much it can be gain by running

way from its vanillas version. Especially for the Tanh kernel, its tests results seem to gain the most.

There is clear many future directions to enrich this area of research. One of the direct extension of this research is to change the way the ensemble of the losses is done in the Gradient Descent training. Another area of the investigation is to modify the loss itself to make the kernels learn the whole space of the queries instead of looking it query by query. Finally, one could also analyze how do the theoretical bounds change with CS-MLE Kernel.

Chapter 7

Appendix

```
#-------------------------------------------------------------------
## Fixing the format of the data from 1 qid:13 1:35 2:68 ... 136:72
## to columns -------------------------------------------------------
#-------------------------------------------------------------------

test <- read.delim("mypath\\test.txt",header=F,sep=" ")
test <- as.data.frame(lapply(test,function(x) as.numeric(gsub(".+:([0-9]*)","\\1",x))))

#-----------------------------------------------------------
## Cost-sensitive ListMLE Ranking Approach --------------
#-----------------------------------------------------------

# norm of a vector ------------------------------------
norm_vec <- function(x) sqrt(sum(x^2))

#-----------------------------------------------------------
# check size k for DCG ---------------------------------
# returns a list with corrected data size and DCGg ------


checksizek <- function(query,kappa){

  # check if query is <= k
  k <- kappa
  actualsize <- dim(query)[1]
  if(actualsize>k){
    query <- query[1:k,]
  } else if(actualsize<k) {
    newrows <- data.frame(matrix(c(rep.int(NA,length(query))),
    nrow=(k - actualsize),ncol=length(query)))
```

```
    colnames(newrows) <- colnames(query)
    query <- rbind(query,newrows)

    query[is.na(query)] <-  rep(0,sum(is.na(query)))
  }
  query[,-1] <- apply(query[,-1],2,function(x)
  x/ifelse(norm_vec(x)==0,rep(1,length(x)),norm_vec(x))) #normalizing x
  # DCG
  g <- function(j){
    1 + sum(y>y[j])
  }

  y <- query[,1]
  dcgg <- 0
  for(i in 1:length(y)){
    dcgg <- dcgg + (2^(y[i])-1)/(log2(1+g(i)))
  }

  return(list(query,dcgg))

}

#-------------------------------------------------------------------
# calculate NDCG@k for a query -------------------------------------
#-------------------------------------------------------------------

nDCG_k <- function(response,k){
  y <- response[1:k]
  dcgg <- 0
  for(i in 1:length(y)){
    dcgg <- dcgg + (2^(y[i])-1)/(log2(1+i))#g(i)
  }

  y <- sort(response[1:k],decreasing = T)
  maxdcgg <- 0
  for(i in 1:length(y)){
    maxdcgg <- maxdcgg + (2^(y[i])-1)/(log2(1+i))
  }
  return(ifelse(maxdcgg==0,0,dcgg/maxdcgg))
}


#-----------------------------------------------------
# calculate grad/hessian for a given query ----------
#-----------------------------------------------------
```

```
grad_listMLE_plain <- function(w,query,dcgg){
  y <- as.matrix(query[,1])
  x <- as.matrix(query[,-1])
  n <- length(y)
  p <- length(w)

  if(sum(y)==0) {m <- matrix(0,nrow = p,ncol = 1)}
  else{ #in case all doc irrelevant

    xw <- x %*% w
    beta <- y/sum(y)

    alpha <- matrix(0,nrow = n,ncol = n)
    for(i in 1:n){
      alpha[i,] <- ifelse(y[i]>y,(y[i]-y)/y[i],0)
    }
    alpha[lower.tri(alpha,diag = T)] <- 0

    grad <- matrix(0,nrow = p,ncol = 2)
    loss <- matrix(0,nrow = n,ncol = 2)
    for(j in 1:p){
      for(i in 1:n){
        # expo <- alpha[i,] * exp(xw - xw[i,] * rep(1,n))
        # expo[1:i] <- 0
        # alpha_exp <- t(expo)  %*% (x - x[i,])
        # loss[i,1] <- beta[i]*alpha_exp/(1 + expo)
        expo <- (exp(xw - xw[i,] * rep(1,n)) * (x - x[i,])[,j])
        expo[1:i] <- 0
        alpha_exp <- alpha[i,] %*% expo
        loss[i,1] <- beta[i]*alpha_exp/(1 + alpha_exp)


        expo2 <- (exp(xw - xw[i,] * rep(1,n)) * ((x - x[i,])[,j])^2)
        expo2[1:i] <- 0
        alpha_exp2 <- alpha[i,] %*% expo2
        loss[i,2] <- beta[i]*alpha_exp2/(1 + alpha_exp)^2
      }
      grad[j,1] <- sum(loss[,1])/(dcgg*log(2)) #gradient
      grad[j,2] <- sum(loss[,2])/(dcgg*log(2)) #hessian
    }

    m <- ifelse(grad[,1]==0 & grad[,2]==0,grad[,1],grad[,1]/grad[,2])
  }
  return(m)
```

```
}

#---------------------------------------------------------
# for a given query do gradient descent and return w ----
#---------------------------------------------------------

i <- j <- 0
w_old <- rep(0.048,136)  # error and w must have as many elements as x variables
w_new <- rep(0.05,136)
error <- rep(0.001,136)
l <- length(error)
grad <- daboius <- NULL
qids <- unique(train$queryID)


while(sum(abs(w_new-w_old) >= error)>0){
  posiids <- 1:length(qids)
  i=i+1
  w_old <- w_new
  for(k in qids){
    j <- j+1;cat("qid: ",k, j,"\n")
    x <- checksizek(train[which(train$queryID == k),-2],1)
    grad <- cbind(grad,grad_listMLE_plain(w_old,x[[1]],x[[2]]))
  }

  w_new <-  w_old - rowSums(grad)/ncol(grad)
  w_new <- w_new/norm_vec(w_new)
  daboius <- cbind(daboius,w_new)
  if (i>=200 | length(qids)==0) break; cat("iteration ",i,"\n")
}
write.csv(daboius,"wplain1.csv",quote = F,row.names = F)

atk_data <- grep("w.*csv",list.files(),value=T)
atk_data <- atk_data[grep("wplain.*csv",atk_data)]
avg_nDCG_polynomial <- matrix(0,nrow = 2000,ncol = length(atk_data))
atk_metric <- 1
j <- 0
qids <- unique(test$queryID)

for(i in atk_data){
  cat(i,"\n")
  j <- j+1
  daboius <- read.csv(i)
  bestw <- dim(daboius)[2]
  avg_nDCG <- NULL
```

```
  for(k in qids){
    x <- checksizek(test[which(test$queryID == k),-2],
    as.numeric(gsub("w[a-z]+([0-9]+).*\\.csv","\\1",i)))
    avg_nDCG <- c(avg_nDCG,nDCG_k(x[[1]][,1][order(as.matrix(x[[1]][,-1]) %*%
    daboius[,bestw],decreasing = T)],atk_metric))
  }
  avg_nDCG_polynomial[,j] <- avg_nDCG
}
apply(avg_nDCG_polynomial,2,mean)

#-----------------------------------------------------------------------
# calculate grad/hessian for a kernelized query ---------------------
#-----------------------------------------------------------------------

kxxi <- function(x,kernel="polynomial",a=0,b=1,d=1,sigma=1,gama=1){
  x <- as.matrix(x)
  xxt <- x %*% t(x)
  kx <- matrix(0,nrow(xxt),nrow(xxt))

  if (kernel=="polynomial"){
    kx <- apply(xxt,1,function(x) (b*x+a)^d)
    return(kx)
  } else if (kernel == "gaussian"){
    for(j in 1:nrow(x)){
      for (i in 1:nrow(x)){
        kx[i,j] <- exp(-sum((xxt[i,]-xxt[j,])^2)/(2*sigma^2))
      }
    }
    return(kx)
  } else if (kernel == "laplace"){
    for(j in 1:nrow(x)){
      for (i in 1:nrow(x)){
        kx[i,j] <- exp(-gama*sum((xxt[i,]-xxt[j,])^2))
      }
    }
    return(kx)
  } else if (kernel == "tanh"){
    kx <- apply(xxt,1,function(x) tanh(b*x+a))
    return(kx)
  }
}


grad_listMLE_kernel <- function(w,query,dcgg,kernel,...){
  y <- query[,1]
```

```
  x <- query[,-1]
  n <- length(y)
  p <- length(w)

  if(sum(y)==0) {m <- matrix(0,nrow = n,ncol = 1)}
  else{ #in case all doc irrelevant

    kx <- kxxi(x,kernel,...)
    kxw <- matrix(rowSums(kx)*w,n,1)
    beta <- y/sum(y)

    alpha <- matrix(0,nrow = n,ncol = n)
    for(i in 1:n){
      alpha[i,] <- ifelse(y[i]>y,(y[i]-y)/y[i],0)
    }
    alpha[lower.tri(alpha,diag = T)] <- 0

    grad <- matrix(0,nrow = n,ncol = 2)
    loss <- matrix(0,nrow = n,ncol = 2)
    for(j in 1:n){
      for(i in 1:n){
        expo <- (exp(kxw - kxw[i,] * rep(1,n)) * (kx - kx[i,])[,j])
        expo[1:i] <- 0
        alpha_exp <- alpha[i,] %*% expo
        loss[i,1] <- beta[i]*alpha_exp/(1 + alpha_exp)

        expo2 <- (exp(kxw - kxw[i,] * rep(1,n)) * ((kx - kx[i,])[,j])^2)
        expo2[1:i] <- 0
        alpha_exp2 <- alpha[i,] %*% expo2
        loss[i,2] <- beta[i]*alpha_exp2/(1 + alpha_exp)^2
      }
      grad[j,1] <- sum(loss[,1])/(dcgg*log(2)) #gradient
      grad[j,2] <- sum(loss[,2])/(dcgg*log(2)) #hessian
    }

    m <- ifelse(grad[,1]==0 & grad[,2]==0,grad[,1],grad[,1]/grad[,2])
  }
  return(m)
}

atk <- 10
i <- j <- 0
w_old <- rep(0.048,atk)  # if kernel error, w must have as many elements as n obs
w_new <- rep(0.05,atk)
error <- rep(0.001,atk)
```

```
l <- length(error)
grad <- daboius <- NULL
qids <- unique(train$queryID)


while(sum(abs(w_new-w_old) >= error)>0){
  i=i+1
  w_old <- w_new
  for(k in qids){
    j <- j+1;cat("qid: ",k,j,"\n")
    x <- checksizek(train[which(train$queryID == k),-2],atk)
    grad <- cbind(grad,grad_listMLE_kernel(w_old,x[[1]],x[[2]],
    kernel="gaussian",sigma=8.246211))
  }

  w_new <-  w_old - rowSums(grad)/ncol(grad)
  w_new <- 10*w_new/norm_vec(w_new)
  daboius <- cbind(daboius,w_new)
  if (i>=200 | length(qids)==0) break; cat("iteration ",i,"\n")
}
write.csv(daboius,"wgaussian10_sigma8.246211.csv",quote = F,row.names = F)

atk_data <- grep("w.*csv",list.files(),value=T)
atk_data <- atk_data[-grep("wplain.*csv",atk_data)]
avg_nDCG_polynomial <- matrix(0,nrow = 2000,ncol = length(atk_data))
atk_metric <- 10#c(3,5,10)
j <- 0
qids <- unique(test$queryID)

for(i in atk_data){
  cat(i,"\n")
  j <- j+1
  daboius <- read.csv(i)
  bestw <- dim(daboius)[2]
  avg_nDCG <- NULL
  for(k in qids){
    x <- checksizek(test[which(test$queryID == k),-2],
    as.numeric(gsub("w[a-z]+([0-9]+).*\\.csv","\\1",i)))
    avg_nDCG <- c(avg_nDCG,nDCG_k(x[[1]][,1][order(kxxi(x[[1]][,-1],
    gsub("w([a-z]+)[0-9]+.*\\.csv","\\1",i)) %*% daboius[,bestw],
    decreasing = F)],atk_metric))
  }
  avg_nDCG_polynomial[,j] <- avg_nDCG
}
apply(avg_nDCG_polynomial,2,mean)
```

Chapter 8

Bibliography

Balcan, Maria-Florina, Nikhil Bansal, Alina Beygelzimer, Don Coppersmith, John Langford, and Gregory B Sorkin. 2008. "Robust Reductions from Ranking to Classification." *Machine Learning* 72 (1-2). Springer: 139–53.

Boyd, Stephen, and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge university press.

Brodersen, Kay H, Cheng Soon Ong, Klaas E Stephan, and Joachim M Buhmann. 2010. "The Binormal Assumption on Precision-Recall Curves." In *Pattern Recognition (ICPR), 2010 20th International Conference on*, 4263–6. IEEE.

Burges, Chris, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. "Learning to Rank Using Gradient Descent." In *Proceedings of the 22nd International Conference on Machine Learning*, 89–96. ACM.

Burges, Christopher JC. 2010. "From Ranknet to Lambdarank to Lambdamart: An Overview." *Learning* 11: 23–581.

Cao, Zhe, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. "Learning to Rank: From Pairwise Approach to Listwise Approach." In *Proceedings of the 24th International Conference on Machine Learning*, 129–36. ACM.

Chapelle, Olivier, and Yi Chang. 2011. "Yahoo! Learning to Rank Challenge Overview." In *Yahoo! Learning to Rank Challenge*, 1–24.

Cortes, Corinna, and Vladimir Vapnik. 1995. "Support-Vector Networks." *Machine Learning* 20 (3). Springer: 273–97.

Cossock, David, and Tong Zhang. 2008. "Statistical Analysis of Bayes Optimal Subset Ranking." *Information Theory, IEEE Transactions on* 54 (11). IEEE: 5140–54.

Crammer, Koby, Yoram Singer, and others. 2001. "Pranking with Ranking." In *Nips*, 1:641–47.

Dahl, David B. 2016. *Xtable: Export Tables to LaTeX or HTML*. http://CRAN.R-project.org/package=xtable.

Dang, V. n.d. "The Lemur Project-Wiki-RankLib." *Lemur Project,[Online]. Available:*

Http://sourceforge. Net/p/lemur/wiki/RankLib.

Franc, Vojtech, and Václav Hlavác. 2004. "Statistical Pattern Recognition Toolbox for Matlab." *Prague, Czech: Center for Machine Perception, Czech Technical University*.

Herbrich, Ralf, Thore Graepel, and Klaus Obermayer. 1999. "Large Margin Rank Boundaries for Ordinal Regression." *Advances in Neural Information Processing Systems*. MIT; 1998, 115–32.

Hiton, Geoffrey. 2012. "6-3-The Momentum Method." University of Toronto: Coursera. www.coursera.org/course/neuralnets.

Järvelin, Kalervo, and Jaana Kekäläinen. 2002. "Cumulated Gain-Based Evaluation of IR Techniques." *ACM Transactions on Information Systems (TOIS)* 20 (4). ACM: 422–46.

Joachims, Thorsten. 2006. "Training Linear SVMs in Linear Time." In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 217–26. ACM.

———. n.d. "SVMrank Support Vector Machine for Ranking." *SVMrank,[Online]. Available: Http://www.cs.cornell.edu/People/tj/svm_light/svm_rank.html*.

Karatzoglou, Alexandros, Alex Smola, Kurt Hornik, and Achim Zeileis. 2004. "Kernlab – an S4 Package for Kernel Methods in R." *Journal of Statistical Software* 11 (9): 1–20. http://www.jstatsoft.org/v11/i09/.

Kiwiel, Krzysztof C. 2001. "Convergence and Efficiency of Subgradient Methods for Quasiconvex Minimization." *Mathematical Programming* 90 (1). Springer: 1–25.

Lan, Yanyan, Tie Yan Liu, Zhiming Ma, and Hang Li. 2009. "Generalization Analysis of Listwise Learning-to-Rank Algorithms." In *Proceedings of the 26th Annual International Conference on Machine Learning*, 577–84. ACM.

Lan, Yanyan, Tie-Yan Liu, Zhiming Ma, and Hang Li. 2008. "Generalization Analysis of Listwise Learning-to-Rank Algorithms Using Rademacher Average." Citeseer.

Lan, Yanyan, Tie-Yan Liu, Tao Qin, Zhiming Ma, and Hang Li. 2008. "Query-Level Stability and Generalization in Learning to Rank." In *Proceedings of the 25th International Conference on Machine Learning*, 512–19. ACM.

Lashkari, Arash Habibi, Fereshteh Mahdavi, and Vahid Ghomi. 2009. "A Boolean Model in Information Retrieval for Search Engines." In *Information Management and Engineering, 2009. ICIME'09. International Conference on*, 385–89. IEEE.

Li, Ping, Qiang Wu, and Christopher J Burges. 2007. "Mcrank: Learning to Rank Using Multiple Classification and Gradient Boosting." In *Advances in Neural Information Processing Systems*, 897–904.

Lin, Hsuan-Tien, and Ling Li. 2012. "Reduction from Cost-Sensitive Ordinal Ranking to Weighted Binary Classification." *Neural Computation* 24 (5). MIT Press: 1329–67.

Liu Tie-Yan. 2008. "Learning to Rank for Information Retrieval." Tutorial. Singapore. http://research.microsoft.com/en-us/people/tyliu/letor-tutorial-sigir08.pdf.

Liu, Tie-Yan. 2009. "Learning to Rank for Information Retrieval." *Foundations and Trends*

in Information Retrieval 3 (3). Now Publishers Inc.: 225–331.

Liu, Tie-Yan, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. 2007. "Letor: Benchmark Dataset for Research on Learning to Rank for Information Retrieval." In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 3–10.

Lu, Min, MaoQiang Xie, Yang Wang, Jie Liu, and YaLou Huang. 2010. "Cost-Sensitive Listwise Ranking Approach." In *Advances in Knowledge Discovery and Data Mining*, 358–66. Springer.

Manning, Christopher D, Prabhakar Raghavan, Hinrich Schütze, and others. 2008. *Introduction to Information Retrieval*. Vol. 1. 1. Cambridge university press Cambridge.

Mendelson, Shahar. 2002. "Rademacher Averages and Phase Transitions in Glivenko-Cantelli Classes." *Information Theory, IEEE Transactions on* 48 (1). IEEE: 251–63.

Meyer, David, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. 2015. *E1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*. http://CRAN.R-project.org/package=e1071.

Page, Lawrence, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. "The PageRank Citation Ranking: Bringing Order to the Web." Stanford InfoLab.

Qin, Tao, Tie-Yan Liu, Wenkui Ding, Jun Xu, and Hang Li. n.d. "Microsoft Learning to Rank Datasets." http://research.microsoft.com/en-us/projects/mslr/.

Qin, Tao, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. "LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval." *Information Retrieval* 13 (4). Springer: 346–74.

Qin, Tao, Xu-Dong Zhang, Ming-Feng Tsai, De-Sheng Wang, Tie-Yan Liu, and Hang Li. 2008. "Query-Level Loss Functions for Information Retrieval." *Information Processing & Management* 44 (2). Elsevier: 838–55.

R Core Team. 2015. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Robertson, Stephen E, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, and others. 1995. "Okapi at TREC-3." *NIST SPECIAL PUBLICATION SP* 109. NATIONAL INSTIUTE OF STANDARDS & TECHNOLOGY: 109.

Salton, Gerard, and Christopher Buckley. 1988. "Term-Weighting Approaches in Automatic Text Retrieval." *Information Processing & Management* 24 (5). Elsevier: 513–23.

Salton, Gerard, Anita Wong, and Chung-Shu Yang. 1975. "A Vector Space Model for Automatic Indexing." *Communications of the ACM* 18 (11). ACM: 613–20.

Shawe-Taylor, John, and Nello Cristianini. 2004. *Kernel Methods for Pattern Analysis*. Cambridge university press.

Sidorov, Grigori, Alexander Gelbukh, Helena Gómez-Adorno, and David Pinto. 2014. "Soft Similarity and Soft Cosine Measure: Similarity of Features in Vector Space Model." *Computación Y Sistemas* 18 (3). Centro de Investigación en Computación, IPN: 491–504.

Snyman, Jan. 2005. *Practical Mathematical Optimization: An Introduction to Basic*

Optimization Theory and Classical and New Gradient-Based Algorithms. Vol. 97. Springer Science & Business Media.

Sparck Jones, Karen. 1972. "A Statistical Interpretation of Term Specificity and Its Application in Retrieval." *Journal of Documentation* 28 (1). MCB UP Ltd: 11–21.

Stigler, Stephen M. 1974. "Gergonne's 1815 Paper on the Design and Analysis of Polynomial Regression Experiments." *Historia Mathematica* 1 (4). Elsevier: 431–39.

Voorhees, Ellen M, and others. 1999. "The TREC-8 Question Answering Track Report." In *Trec*, 99:77–82.

Wickham, Hadley. 2009. *Ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York. http://ggplot2.org.

Wickham, Hadley, and Romain Francois. 2015. *Dplyr: A Grammar of Data Manipulation*. http://CRAN.R-project.org/package=dplyr.

Wickham, and Hadley. 2007. "Reshaping Data with the Reshape Package." *Journal of Statistical Software* 21 (12). http://www.jstatsoft.org/v21/i12/paper.

Xia, Fen, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. "Listwise Approach to Learning to Rank: Theory and Algorithm." In *Proceedings of the 25th International Conference on Machine Learning*, 1192–9. ACM.

Xie, Yihui. 2013. "Knitr: A General-Purpose Tool for Dynamic Report Generation in R." *R Package Version* 1 (1).

———. 2014. "Knitr: A Comprehensive Tool for Reproducible Research in R." *Implement Reprod Res* 1: 20.

———. 2015. *Dynamic Documents with R and Knitr*. Vol. 29. CRC Press.

Xu, Jun, and Hang Li. 2007. "Adarank: A Boosting Algorithm for Information Retrieval." In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 391–98. ACM.