Efficient Exploration of Gradient Space of Online Learning to Rank

Masters Thesis
University of Virginia

Sonwoo Kim

Spring 2018

1 Abstract

Learning to rank is an optimization problem to update the ranking function to score each document to rank them. As opposed to offline learning to rank, which used annotated data, online learning to rank utilized the abundant click data users create. While vector spaced exploration based online learning to rank algorithm has shown improvements in recent works, they are still limited to uniform exploration, and ineffective comparison of candidate rankers' performance. A new algorithm, Null Space Gradient Descent is presented. Its contributions are in three parts: 1) null space exploration to prevent historically bad gradient direction, 2) candidate preselection to choose candidates that produce different ranking order in a given query, 3) effective tie breaking technique to use historically difficult queries. Extensive experiments show that NSGD's NDCG performance is stronger than most of baselines and converges much faster.

2 Introduction

Learning to rank (L2R) is a ranking problem to find best combination of various information retrieval features in relation to a given query. With abundance of online documents available, how to rank documents to maximize user satisfaction is becoming more important than ever. This Masters thesis will explain different types of learning to rank algorithms, their limitations, and present a research work that outperforms state-of-the-arts algorithms. This research was done as a part of Professor Hongning Wang's group project, and much of the work will be published.

3 Learning to Rank

To rank documents by relevance to the given query. There are many criterion presented from the field of information retrieval (IR). Many of them measure how often the given document keyword matching occurs to the query in comparison with other documents. Many measurements have different formulae, and there are also characteristics of the document that are often associated with its relevance. Figuring out the best combination, how much each criterion contributes to judging a document's relevance, is the problem of learning to rank. The characteristics and IR measurements of each document in comparison to the query is referred as "features" of the document, which are often provided as a vector.

In a simple linear model, in which documents are ranked by the dot product of the feature and weight vectors, the weight vector is the combination of how each feature contributes to relevance judgment. Figure 1 provides an illustration of this.

The traditional way of learning to rank was done with annotated data, which we call an "offline" fashion. Datasets, in which each documents relevance labeled by trained professionals, are used to train weight vectors, using different machine learning techniques. While such approach has been successful, it contains several limitations. First, manually annotated data is expensive so its size is limited. Second, the editor's annotation cannot always be trusted as the absolute truth, and may not align with every user preference. Lastly, the user preference often changes dynamically and it is hard for the ranker to adapt to changed user preference if it relies too much on old datasets.

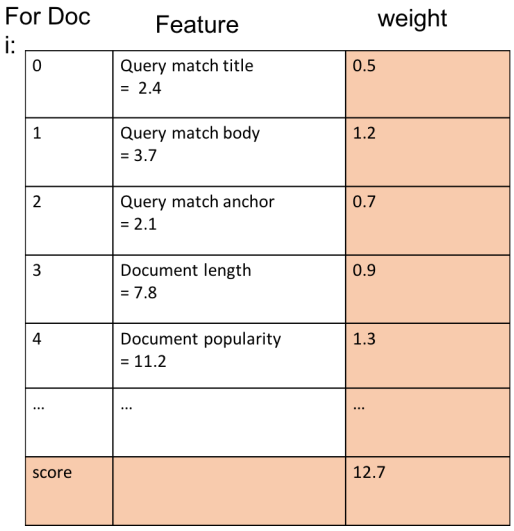


Figure 1: How ranking is computed in Linear Model

To overcome such limitations, online learning to rank has been given much attention. Instead of using scarce manually annotated data, online L2R algorithms utilize feedback users create, particularly what documents they click. And the algorithm will train its ranker "on the fly" as it collects more data from user clicks and directly exploits user feedback in real time. Users automatically generate such feedback as they use the system, so the algorithm can make use of the abundance of data from actual users it tries to satisfy.

Still, user's click data has limitations in nature. Most users in real world do not read through the ranked list from beginning to end. They are more likely to read only documents in top few positions and decide to leave the application whether they are satisfied or not. Because only top few documents get chance to be evaluated by user, if a document did not receive click, the algorithm cannot tell if the user did not find it relevant, or whether it was even looked at (position bias, [4]). Also, even when users look at same results, not every user clicks on same set of documents. Each individual user's judgment differs to some extent (click data is noisy). Online learning to rank algorithms have been developed overcome such limitations, while maximizing its advantage.

While user's click through feedback cannot tell a document's absolute relevance judgment, it can tell relative judgment, comparing against different rankers, or whether a clicked document is more relevant than unclicked (skipped) one in upper position.

For consistent comparison among learning to rank algorithms, most online learning to rank research works use the same datasets but simulate user behavior to collect click feedbacks for judgement, instead of using absolute label. Experiments even introduce click noise in this process. Due to their limitations in judgment collection, when the same datasets were given, online learning to rank algorithms still perform worse than offline versions (online algorithms will have a lot more training data in real life). While my research focused on online learning to rank algorithms, I also compared them against offline counterparts.

4 Basic Online Learning to Rank

Among many common approaches to online learning to rank problem, my research focused on vector space model. It sets the weight as a vector in n-dimensional space (n being the dimension of the feature vector) and try to find the optimal weight, w*. In every iteration, the algorithm proposes variation to current weight vector (gradients, u_t^n), which becomes candidate rankers by $w_t + u_t^n$. Their performance is compared, and if any of proposed ranker performs better than current ranker, w_t, it will be updated. Figure 2 provides the overview of such algorithms. One representative work in this approach is Dueling Bandit Gradient Descent (DBGD) [10]. As figure 3 illustrates, DBGD takes one exploratory ranker by uniformly sampling gradient (u_t) to the weight vector.Like mentioned earlier, the exploratory ranker ($w_t + u_t$) and current ranker (w_t) is

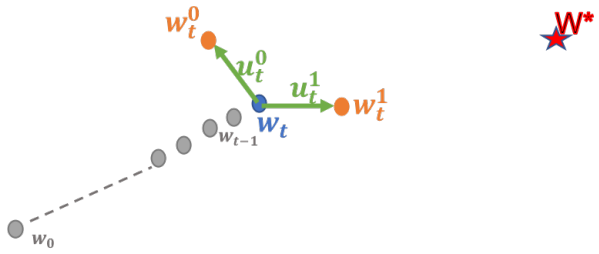


Figure 2: Overview of Vector Space Model

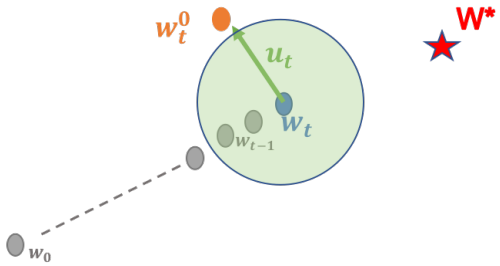


Figure 3: Overview of DBGD

compared in performance, and current ranker is updated if exploratory one outperforms ($w_{t+1} = w_t + u_t$).

There has been many variation to the DBGD. Instead of limiting single exploratory direction, the idea of multiple exploratory direction is proposed in Multileave Gradient Descent algorithm [9]. Instead of sampling directions purely uniformly, two gradients in opposite direction to each other is proposed and this algorithm, the Dual-Point Dueling Bandit Gradient Descent algorithm [11] also outperformed DBGD. Figures 4 and 5 illustrates the exploratory methods of Multileave and Dualpoint respectively.

Despite their improvements over DBGD, both and many more variations to DBGD has important limitations. First, they propose gradient from uniform sampling. Uniform sampling does not take historical performance of gradient directions into account, and leads to repetitive exploration which has performed bad historically, and therefore ineffective gradient exploration. Figure 6 shows how uniform sampling could keep proposing directions away from optimal weight, $w*$. Second, while proposed gradients may differ in their original vector form, they do not take the current query into account. Because feature vector (characteristic of a document in relation to the given query) is dependent to the current query, independence to the current query may lead to the same order of ranking scores (dot product of weight and feature vectors). If the ranking order is the same, the algorithm cannot differentiate the performance of the rankers. Figure 7 shows the problem of query independence in candidate selection.

5 Null Space Gradient Descent (NSGD)

To overcome these problems, my group has proposed a new algorithm, Null Space Gradient Descent (NSGD). Its novelty can be summarized into three ideas and they are:

1. Sample proposal directions from the null space of previously explored gradients that performed poorly.
2. Preselect proposed rankers with focus on giving different ranker orders
3. Compare tied rankers on historically worst performing queries to break ties.

The following sections will explain each component in more detail.

5.1 Null Space

In order to prevent repetitive exploration to historically bad directions, the NSGD algorithm maintains a queue of recently explored gradients and its performance (received click). NSGD has a fixed window size to

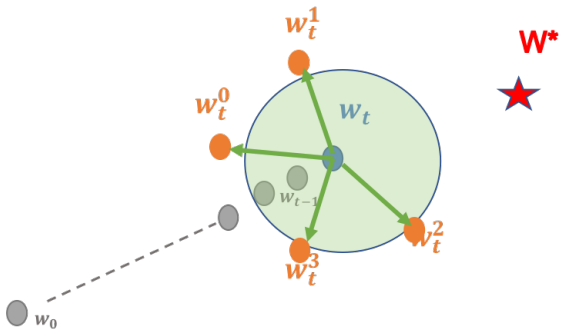


Figure 4: Overview of Multileave

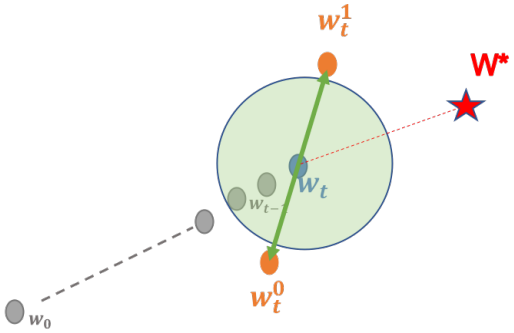


Figure 5: Overview of DualPoint

take gradients of which the performance has been measured in past N iterations. This was done to prevent completely excluding exploration to directions that performed badly much earlier. From this queue Q_g, top k worst performing historical directions are chosen, $G[g^1, g^2, g^3, ...]$. Then new exploratory directions are sampled from the null space of the chosen gradients, G. This prevents repeated exploration in ineffective directions in G. Figure 8 shows why sampling from the null space is effective.

The fixed window size (how recently the gradient must have been explored), as well as selection size (how many worst gradients are kept) are some of the hyper-parameters that will be further experimented.

5.2 Context Dependent Candidate Preselection

We want to minimize the chance of having proposed candidate rankers result in same exact ranking order with the current ranker, w_t. But a very distinct ranking indicates a higher risk of worse result quality, especially at later stages (complete opposite order of documents is the most different ranking order, but it is likely to perform bad). To balance the risk of both and adequately select candidates that is likely to perform well, NSGD selects top m gradients that maximize the product $|\bar{X} \cdot g_t^i|$.

The intuition behind $|\bar{X} \cdot g_t^i|$ follows: for ranked lists to differ, at least one document has different ranking score under both rankers, and scores are computed as $x \cdot w_t$.

$$\begin{aligned} \exists j, |x_j \cdot (w_t^i - w_t^0)| > 0 \\ \exists j, |x_j \cdot g_t^i| > 0 \\ |\bar{X} \cdot g_t^i| > 0 \end{aligned} \tag{1}$$

While maximization of the dot product of average feature vector to the gradient does not completely guarantee different ranking order, it significantly improves the change they will differ. Figure 9 shows how

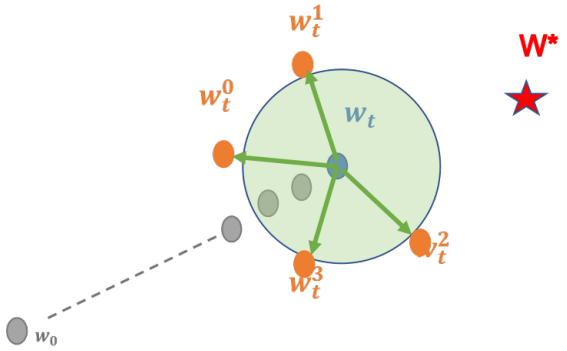


Figure 6: Problem of Uniform Sampling

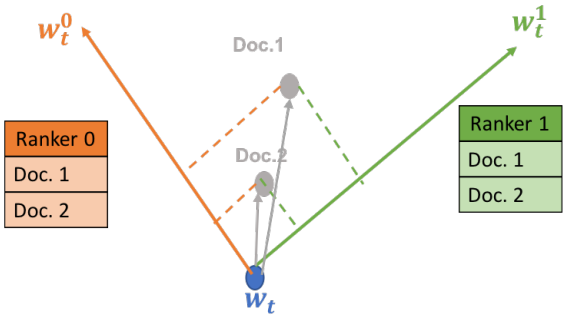


Figure 7: Problem of Common Ranking

context dependent candidate preselection (CDP) is effective in pre-selecting rankers that differ in resulted ranked list.

5.3 Tie Breaking

Even with different ranking orders results from current and exploratory rankers, when comparing the performance between the rankers, ties often occur. To understand what "tie" actually means and why ties occur so often, the concept of intereaving is discussed below.

Interleaving is a method to evaluate ranker's performance by comparing click data of different rankers. It basically combines multiple rankers by inserting one another, or "interleaving". Then once user click is collected from the interleaved ranked list, the winning ranker is determined. Variations of how to interleave and how to assign credit for each clicked document exists, but one of basic, most common form of interleaving is Team Draft Interleaving [6]. Similar to the common Team Draft methodology where coin is tossed to pick a team to give first chance to pick a player when there are equal number of players for each team, Team Draft Interleaving forms the combined ranked list. If the document one ranker is already picked to the combined list, ranker will automatically choose the next document in its ranking. The original ranker that originally placed the document into the combined list is stored, so that after user click is collected, the credit is given solely to that ranker. Figure 10 depicts the process of Team Draft Interleaving. When there are more than two rankers to be combined and compared, multi-interleaving [9] is used, same methodology except multiple rankers take their turns in placing documents.

A problem with such interleaving with multiple rankers is the frequency in which ties occur. Ties occur when the candidate rankers that form the combined list gets the same number of clicks. As mentioned above, users only bother to look at documents positioned at top few positions and click even fewer. This results in scarcity of the clicks when it comes to click-credit comparison among candidate rankers, and especially when there are more rankers to compare, it is likely each ranker will only have one or two clicks collected. This

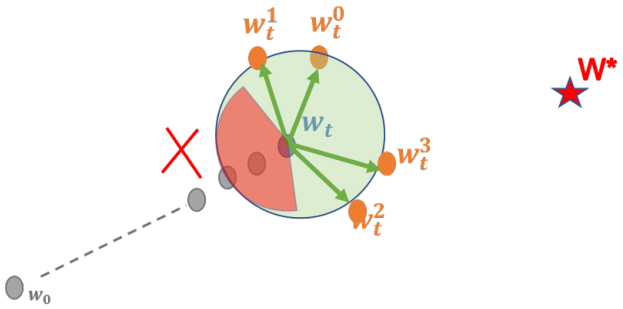


Figure 8: Null Space exploration

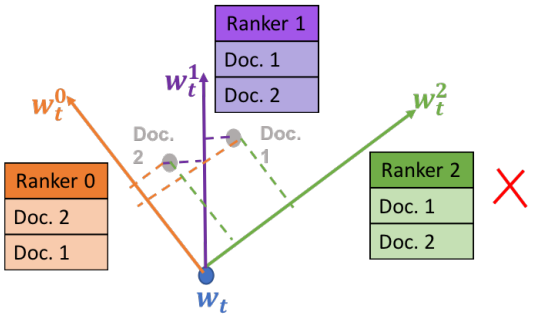


Figure 9: Context Dependent Candidate Preselection

results in high occurence of ties in which the algorithm cannot differentiate which ranker is better. Most current algorithms handle ties arbitrarily, such as simple coin toss or making mean vector of all winning rankers' gradients.

The proposed NSGD break ties more efficiently, by comparing the tied rankers' performance in historical queries. NSGD maintains a collection of most difficult queries in recent iterations, and the clicked documents from those queries. Each winning ranker will rank documents for each of the difficult queries, and their performance of how high each ranker placed the previously clicked document is recorded. The ranker with the highest mean score from the whole list of difficult queries is chosen as the winner. Figure 11 illustrates ties is broken in NSGD through historical difficult queries.

Figure 12 and algorithm 1, summarize the three main components of the NSGD algorithm.

6 Experiments

In the experiment section, performance of NSGD algorithm is compared against baselines in common datasets in learning to rank community. Then more detailed experiment of NSGD is conducted to understand good performance of NSGD.

6.1 Experiment Setup

• **Datasets.** The 5 benchmark datasets used in the experiment are parts of the LETOR 3.0 and LETOR 4.0 collections [5]. Some (NP2003, HP2003) are from navigational tasks, such as home page finding, while TD2003 is from informational tasks, such finding information about specific query. Others (MQ2007, MQ2008) are combinations of two tasks. Informational tasks tend to be difficult for rankers to satisfy user needs. Each dataset contains query id, doc id, relevance label, and list of feature values (differing in size for each dataset). Each dataset is divided into 5 folds for training and testing sets.

• **Click Simuation.** As mentioned above, online learning to rank algorithms simulate user behavior when using annotated datasets. Users tend to lend to leave after browsing first few documents, and their clicks are

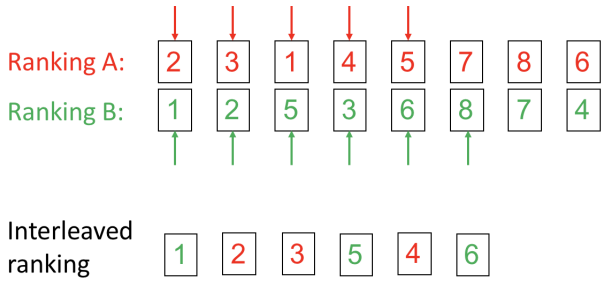


Figure 10: Team Draft Interleaving

Query	Clicked docs
Q35	doc11, doc15, doc7
Q47	doc3, doc10, doc5
Q51	doc4, … , doc5
Q56	…

Query	ranker 0	ranker 1	ranker 3
Q35	0.2	0.3	0.1
Q47	0.32	0.4	0.4
Q51	0.1	0.4	0.3
Q56	…	…	…
Avg.	0.31	0.42 H	0.28

Figure 11: Tie Breaking from Historically Difficult Queries

noisy. We use the commonly use Cascade Click Model [2], which simulates when user clicks, stops looking at documents, when encountering each document's label. Documents with higher relevance label are more likely to trigger user to click on it, and also have higher chance the user will leave after reading the document. There are three types of user behavior, depending on what types of taks user want to perform. Perfect user behaves in controlled manner, closely following the label. Navigational has some noise, and Informational model has more. The simulation of Casscade Click Model's user behavior is summarized in 1. The whole experiment of both baselines and NSGD was conducted in Lerot [7], an online learning to rank framework that automates the process of training and evaluating and contains much of published online learning to rank algorithm implementations.

• **Evaluation Metrics.** The performance of different online learning to rank algorithms are measured in Normalized Discount Cumulative Gain (NDCG). NDCG is the metric that especially promotes rankers that puts relevant documents on upper positions, to meet the position bias many learnin to rank algorithm face in real life. Both offline and online NDCG is computed. While offline NDCG is computed from test queries for each training iteration, online NDCG is the cumulative (with discount factor of 0.995) NDCG value of training queries. It reflects the performance of the algorithms while users interact with the system for training. Training performance, especially in the earlier iterations are important, because it keeps the users satisfied attract them to keep using it, generating more click data. To compare online NSGD's performance against offline algorith, LambdaRank [1] is used.

• **Baselines** Four popular algorithms are used as baselines. Each of them are based on previously mentioned DBGD algorithm, with couple variations. They are summarized below:

- **DBGD** [10]: A single uniformly sampled gradient. Team Draft is used to interleave the results of current and exploratory rankers for comparison.
- **CPS** [3]: A candidate preselection strategy that uses historical data to preselect (best performing) the proposed rankers before the interleaved test in DBGD.
- **DP-DBGD** [11]: Two opposite uniformly sampled directions are explored in DBGD. Both Contextual Interleave, which favors the winning direction from the previous iteration, and Team Draft are used in it in our experiment.

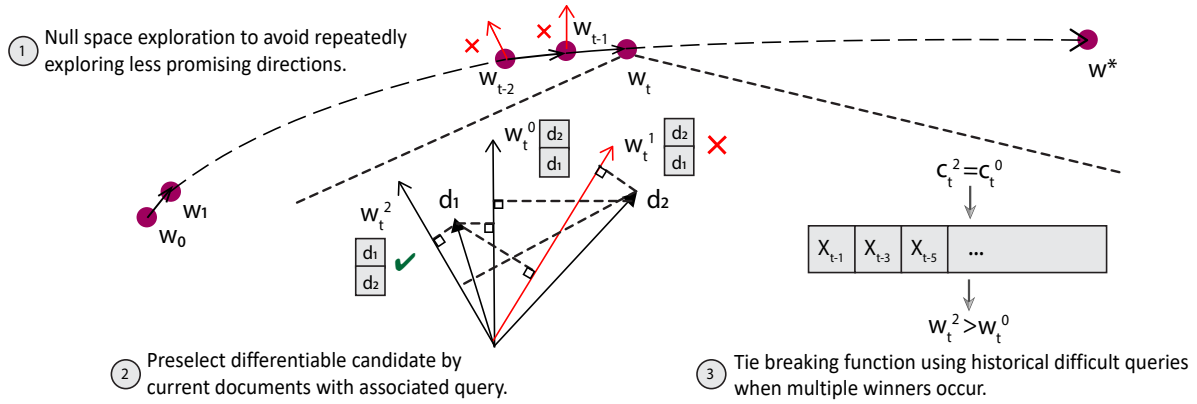


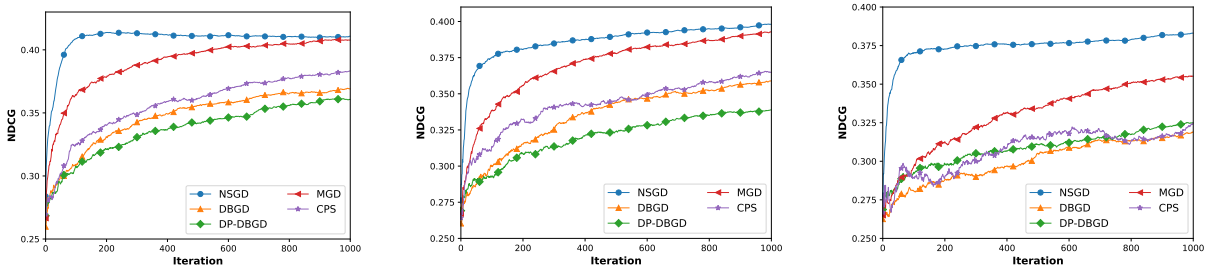
Figure 12: Overview of NSGD algorithm

	Click Probability			Stop Probability		
Relevance grade	0	1	2	0	1	2
Perfect	0.0	0.5	1.0	0.0	0.0	0.0
Navigational	0.05	0.5	0.95	0.2	0.5	0.9
Informational	0.4	0.7	0.9	0.1	0.3	0.5

Table 1: Configurations of simulation click models.

- **MGD** [8]: Multiple uniformly sampled directions are explored in each iteration. Multileave is used to compare performance of each candidate rankers. If there is a tie, the model updates towards the mean of all winners.

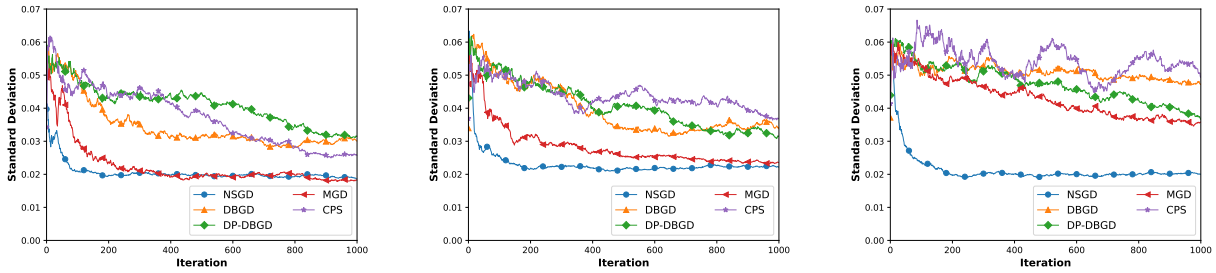
6.2 NDCG Performance Comparison in All Datasets



(a) Perfect (b) Navigational (c) Informational

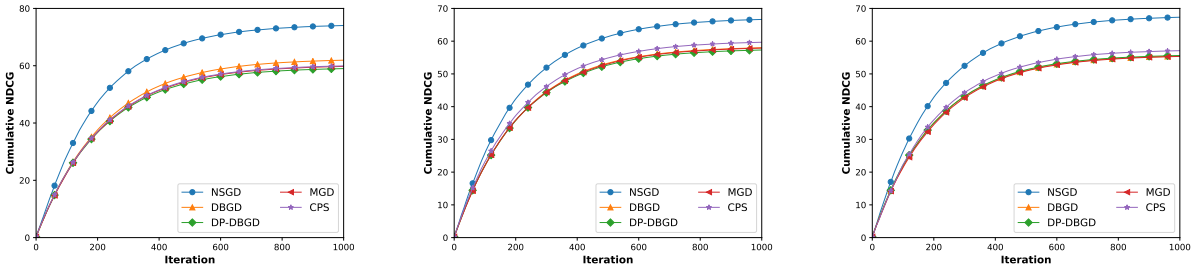
Figure 13: Offline NDCG@10 on MQ2007 dataset under three click models.

In this section, both offline (Table 3)and and online (Table 2) NDCG performance is compared in the 5 datasets, for each of three click models (Perfect, Navigatioal, Informational). Each experiment was run 15 times for each fold, so each values in the summary table below is the mean of 75 runs. Standard deviation of the results is measured for how much results vary, and statistical t-test is performed to verify whether NSGD's outperformance is statistically meaningful. In most datasets, NSGD was the best performing algorithm, especially in the informational click model, where there is more noise. What is more astonishing is the performance in the online NDCG. The results in Table 2 show how clearly NSGD ourperforms all baselines in any dataset and click models. Figures 13, 14, and 15 are graphs of the performance in MQ2007, Informatioal click model. In both Figures 13 and Figure 15 NSGD had better final performance, converge much faster, and even had the loweest variance in the results (from very early iterations), as in Figure 14.



(a) Perfect (b) Navigational (c) Informational

Figure 14: Standard deviation of offline NDCG@10 on MQ2007 dataset under three click models.



(a) Perfect (b) Navigational (c) Informational

Figure 15: Discounted cumulative NDCG@10 on MQ2007 dataset under three click models.

6.3 Detailed Experiment

This section further examines the performance of NSGD in more detail. Here are the questions we want to answer:

1. Is NSGD's exploration more efficient than uniform sampling from the entire parameter space?
2. How do the different components in NSGD contribute to its final performance?
3. How do different settings of hyper-parameters alter the performance of NSGD?

• Cosine similarity with offline model, w*.

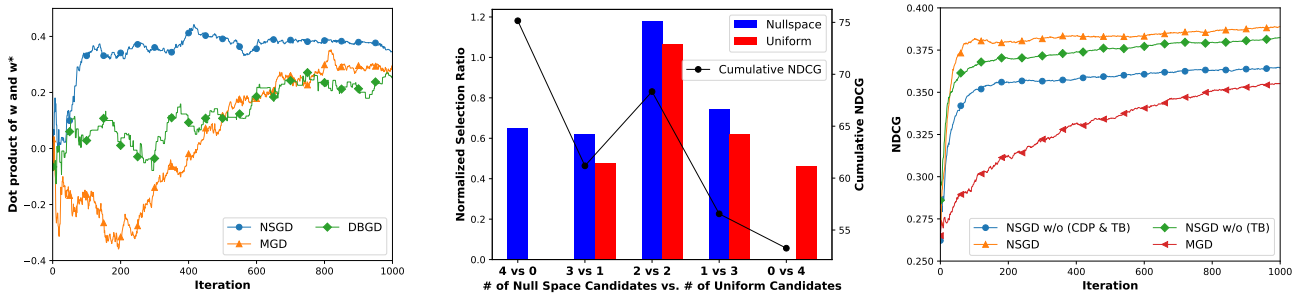
To answer the first question, whether NSGD's exploration more efficient than uniform sampling, We compared the trained weight vectors of NSGD and baseline algorithms against that of offline Lambdarank algorithm. As mentioned, offline algorithms take full usage of every document label, as it does not simulate user behavior and therefore does not suffer from position bias. So, in the same dataset, Lambdarank performs better than all of online learning to rank algorithms. Setting the weight vector from Lambdamart as the optimal weight vector, $w*$, we computed cosine similarity of online algorithms to see how fast each of them converge to the optimal weight vector. As can be seen in Figure 16a, NSGD converges to $w*$ much faster than baselines.

• Selection Ratio of Null Space Exploration

To further examine the efficiency of null space exploration, candidate rankers from both uniform and null space exploration are added, then compared which rankers wins the interleaving comparison. Setting the total number of exploratory rankers to 4, we varied the proportion of null space exploration (from 0 to 4), the the rest as uniform. Then the selection (winning the interelave test and replacing the current weight vector, w_t. Then the selection frequency is normalized by the number of rankers existing to balance out the number of each type of exploratory rankers. As Figure refFig:detailb suggests, null space exploratory rankers (blue) always have higher selection ratio, no matter how many there are (after normalization). This further confirms the efficiency of the null space exploration.

• Ablation Analysis

To answer the second question, how do the different components in NSGD contribute to its final performance, ablation analysis is conducted. Starting from regular DBGD baseline, each of the three components is added one by one. In figure refFig:detailc, the contribution of each component can be measured by the

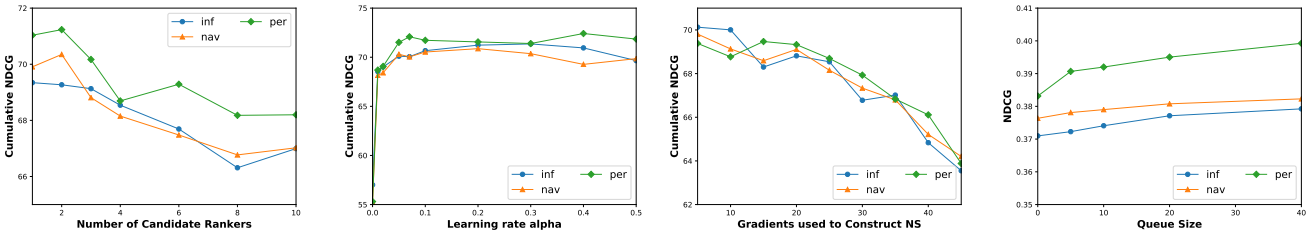


(a) Cosine similarity between online learnt model and offline best model w^*

(b) Selection ratio comparing null space and uniform gradients

(c) Ablation analysis of NSGD

Figure 16: Detailed experimental analysis of NSGD.



(a) Number of candidates

(b) Learning rate α

(c) Number of historical gradients to construct null space

(d) Number of historical queries for tie breaker

Figure 17: Performance of NSGD under different hyperparameter settings

performance gap between each graph. As Figure refFig:detailc suggests, the contribution to performance is in order of:

NullSpace > Preselection > Tie-breaking

• **Hyperparameter Analysis**

The affects of different hyperparamers are measured and compard. The four hyperparameters to be analized are: 1) number of candidate rankers, 2) learning rate, 3) number of historical gradients to construct null space, and 4) number of historical queries for tie breaker. The resulting data is summarized in Figure 17.

1) Number of Candidate Rankers

We varied the number of exploratory candidate rankers, from 1 to 10. While it seems the more candidiate rankers to explore, the better it would be, excessive exploration introduces noise and sparsity of clicks from user becomes worse as each ranker has to share the given number of clicks. Too many candidate rankers hurt appropriate comparison of those candidate rankers. As it turns out in 17a, 2 to 4 candidate rankers are most effective.

2) Learning Rate

Learning rate, *alpha*, is how much the weight vector, w, will be updated by the gradient. The higher the value is, more aggressive the exploration will be. And learning too aggressively often results in worse performance, as it often passes the optimal point. As it turns out in Figure 17b, the performance flattens out around $alpha = 0.005$, and it is the value used in the rest of the work.

3) Number of Historical Gradients to Construct Null Space This is how many worst historical gradients to to create null space. In the MQ2007 dataset this experiment is done, the feature dimension is 46, so we could vary this parameter from 1 to 40 (it cannot be larger the feature dimension). While using more vectors to create the null space may prevent unnecessary exploration into bad direction more aggressively, having too many vectors often limit exploration into a good direction. As in Figure 17c, the decrease in performance

from increasing the number of gradient vectors appeared very soon. After 25 gradients, the performance decrease become more noticeable. The reason why it performed the best at 1 gradient, and there was no increase trend is to be studied further.

4) Number of Historical Queries for Tie Breaker Lastly, we varied the number of queue size of historically difficult queries in tie breaking. As apposed to increasing vectors to construct null space, increase the number of these difficult queries does not limit anything that affects NDCG performance, it will only increase the accuracy of the tie breaking. The trend also shows in Figure 17d, where there is no decrease trend. But after some point, increasing the number of difficult queries will not help much. It is possible increasing the query size may introduce bias, as the earlier queries tend to be recorded as "difficult" because the algorithm keeps on improving. But from the trend in the graph, it seems the effect of this bias is limited as having greater number of difficult queries limit the effect of earlier queries as it becomes diluted.

7 Conclusion

In past three semester as UVA's Masters student, I was exposed to various knowledge in the field of information retrieval. The research started from implementing a learning to rank system. In this thesis, I explained what a learning to rank problem is, two types of learning to rank problems (offline vs online), their limitations, and why online learning to rank has received more attention. A simple vector-space-model based online learning to rank algorithm (DBGD) is explained, their variations, as well as their limitations despite their improvements. Then a new algorithm to overcome such limitation, Null Space Gradient Descent (NSGD) is presented. Three main contribution of this algorithm is summarized, and its experiment results, both performance comparison with common baselines as well as deeper analysis of what leads to its effectiveness (higher ranking performance and faster improvement).

References

[1] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[2] Fan Guo, Chao Liu, and Yi Min Wang. Efficient multiple-click models in web search. In *Proceedings of the Second ACM International Conference on WSDM*, pages 124–131. ACM, 2009.

[3] Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval. *Information Retrieval*, 16(1):63–90, 2013.

[4] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *ACM SIGIR Forum*, volume 51, pages 4–11. Acm, 2017.

[5] Tie-Yan Liu, Jun Xu, Tao Qin, Wenying Xiong, and Hang Li. 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*, volume 310, 2007.

[6] Filip Radlinski, Madhu Kurup, and Thorsten Joachims. How does clickthrough data reflect retrieval quality? In *Proceedings of the 17th ACM CIKM*, pages 43–52. ACM, 2008.

[7] Anne Schuth, Katja Hofmann, Shimon Whiteson, and Maarten de Rijke. Lerot: An online learning to rank framework. In *Proceedings of the 2013 workshop on Living labs for information retrieval evaluation*, pages 23–26. ACM, 2013.

[8] Anne Schuth, Harrie Oosterhuis, Shimon Whiteson, and Maarten de Rijke. Multileave gradient descent for fast online learning to rank. In *Proceedings of the Ninth ACM International Conference on WSDM*, pages 457–466. ACM, 2016.

[9] Anne Schuth, Floor Sietsma, Shimon Whiteson, Damien Lefortier, and Maarten de Rijke. Multileaved comparisons for fast online evaluation. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 71–80. ACM, 2014.

[10] Yisong Yue and Thorsten Joachims. Interactively optimizing information retrieval systems as a dueling bandits problem. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1201–1208. ACM, 2009.

[11] Tong Zhao and Irwin King. Constructing reliable gradient exploration for online learning to rank. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1643–1652. ACM, 2016.

Algorithm 1 Null Space Gradient Descent (NSGD)

Inputs: $\delta, \alpha, n, m, k_g, k_h, T_g, T_h$
Initiate $w_0^0 = sample\_unit\_vector()$
Set $Q_g = queue(T_g)$ and $Q_h = queue(T_h)$ as fixed size queues
for $t = 1$ to T **do**
    Receive query $X_t = \{x_1, x_2, ..., x_s\}$
    Generate ranked list $l(X_t, w_t^0)$
    $\bar{x}_t = \sum_{i=1}^{s} x_i$
    Construct $= [g^1, ..., g^{k_g}]$ by directions selected from Q_g with the worst recorded quality q
    $^{\perp} = NullSpace()$
    for $i = 1$ to n **do**
        $g_t^i = sample\_unit\_vector(^{\perp})$
    end for
    Select top m gradients that maximize $\left|\bar{x}_t^{g_t^i}\right|$ from $\{g_t^i\}_{i=1}^n$
    for $i = 1$ to m **do**
        $w_t^i = w_t^0 + \delta g_t^i$
        Generate ranked list $l(X_t, w_t^i)$
    end for
    Set $L_t = \text{Multileave}\big(\{l(X_t, w_t^i)\}_{i=0}^m\big)$, and present L_t to user
    Receive click positions C_t on L_t, and infer click credits $\{c_t^i\}_{i=0}^m$ for all rankers
    Infer winner set B_t from $\{c_t^i\}_{i=0}^m$
    if $|B_t| > 1$ **then**
        Select k_h worst performing queries $\big\{(X_i, L_i, C_i)\big\}_{i=1}^{k_h}$ from Q_h by $Eval(L_i, C_i)$.
        $j =_{o \in B_t} \sum_{i=1}^{k_h} Eval(l(X_i, w_o), C_i)$
    else
        Set j to the sole winner in B_t
    end if
    if $j = 0$ **then**
        $w_{t+1}^0 = w_t^0$
    else
        $w_{t+1}^0 = w_t^0 + \alpha g_t^j$
    end if
    for $i = 1$ to m **do**
        $q_t^i = c_t^i - c_t^0$
        if $q_t^i < 0$ **then**
            Append (g_t^i, q_t^i) to Q_g
        end if
    end for
    Append (X_t, L_t, C_t) to Q_h
end for

Table 2: Online score (discounted cumulative NDCG@10) and standard deviation of each algorithm after 1000 queries under each of the three click models. Statistically significant improvements over MGD baseline are indicated by (p¡0.05).

Click Model	Dataset	DBGD	CPS	DP-DBGD	MGD	NSGD
Perfect	**MQ2007**	61.931 (5.535)	59.936 (4.875)	58.995 (4.926)	59.765 (3.015)	**74.038** (3.629)
	MQ2008	81.327 (6.224)	77.694 (6.137)	76.192 (6.452)	77.543 (4.827)	**88.811** (6.022)
	HP2003	110.012 (8.627)	109.279 (8.565)	92.422 (11.358)	101.675 (4.943)	**113.890** (8.276)
	NP2003	101.004 (8.702)	98.774 (8.884)	79.636 (13.338)	104.677 (5.399)	**115.145** (6.287)
	TD2003	39.856 (7.770)	38.054 (6.999)	34.289 (7.703)	38.380 (5.383)	**42.402** (7.654)
Navigational	**MQ2007**	57.989 (4.657)	59.669 (4.911)	57.301 (4.816)	57.884 (3.266)	**66.635** (2.832)
	MQ2008	76.411 (5.983)	75.603 (7.230)	74.984 (5.959)	75.001 (5.085)	**84.091** (4.553)
	HP2003	95.775 (14.394)	95.925 (12.628)	88.773 (11.518)	82.244 (26.944)	**109.783** (5.634)
	NP2003	84.699 (12.275)	88.240 (13.039)	74.521 (14.810)	100.581 (8.962)	**109.433** (5.649)
	TD2003	33.954 (8.368)	35.857 (8.729)	31.468 (7.322)	36.092 (5.616)	**41.274** (7.318)
Informational	**MQ2007**	55.427 (5.639)	57.094 (5.689)	55.619 (5.066)	55.338 (3.395)	**67.312** (3.438)
	MQ2008	73.941 (6.101)	74.825 (5.419)	72.392 (6.259)	72.757 (4.690)	**84.053** (4.980)
	HP2003	59.376 (23.637)	56.004 (22.101)	66.295 (16.782)	75.314 (11.281)	**108.592** (5.503)
	NP2003	56.996 (20.547)	54.615 (19.354)	62.067 (17.667)	74.497 (13.249)	**108.624** (5.831)
	TD2003	23.021 (8.675)	23.826 (7.964)	24.948 (6.848)	28.482 (5.299)	**39.386** (7.148)

Table 3: Offline score (NDCG@10) and standard deviation of each algorithm after 1000 queries under each of the three click models. Statistically significant improvements over MGD baseline are indicated by (p¡0.05).

Click Model	Dataset	DBGD	CPS	DP-DBGD	MGD	NSGD
Perfect	**MQ2007**	0.369 (0.030)	0.383 (0.026)	0.361 (0.032)	0.408 (0.018)	**0.411** (0.019)
	MQ2008	0.465 (0.042)	0.474 (0.042)	0.461 (0.041)	0.487 (0.037)	**0.488** (0.043)
	HP2003	0.760 (0.067)	0.764 (0.068)	0.762 (0.062)	**0.771** (0.062)	0.752 (0.752)
	NP2003	0.704 (0.052)	0.702 (0.050)	0.682 (0.062)	0.712 (0.048)	**0.714** (0.049)
	TD2003	0.267 (0.082)	0.296 (0.094)	0.286 (0.091)	**0.308** (0.096)	0.289 (0.092)
Navigational	**MQ2007**	0.359 (0.034)	0.365 (0.037)	0.339 (0.031)	0.393 (0.024)	**0.398** (0.022)
	MQ2008	0.459 (0.038)	0.456 (0.037)	0.445 (0.045)	0.477 (0.036)	**0.478** (0.037)
	HP2003	0.728 (0.063)	0.734 (0.072)	**0.752** (0.061)	0.707 (0.156)	0.744 (0.073)
	NP2003	0.709 (0.035)	0.661 (0.066)	0.675 (0.061)	0.707 (0.052)	**0.710** (0.039)
	TD2003	0.276 (0.095)	0.285 (0.093)	0.269 (0.087)	**0.303** (0.098)	0.274 (0.094)
Informational	**MQ2007**	0.319 (0.047)	0.325 (0.049)	0.325 (0.037)	0.355 (0.036)	**0.383** (0.020)
	MQ2008	0.425 (0.050)	0.434 (0.047)	0.422 (0.054)	0.450 (0.041)	**0.472** (0.036)
	HP2003	0.500 (0.196)	0.463 (0.191)	0.669 (0.103)	**0.736** (0.063)	0.713 (0.069)
	NP2003	0.526 (0.190)	0.443 (0.179)	0.657 (0.118)	0.660 (0.059)	**0.707** (0.044)
	TD2003	0.174 (0.099)	0.178 (0.092)	0.219 (0.094)	**0.271** (0.090)	0.251 (0.085)