Poisoning attacks on Online Learning to Rank

A

Thesis

Presented to

the faculty of the School of Engineering and Applied Science

University of Virginia

---

in partial fulfillment

of the requirements for the degree

Master of Science

by

Rishab Bamrara

May 2021

APPROVAL SHEET

This

Thesis

is submitted in partial fulfillment of the requirements
for the degree of

Master of Science

Author: Rishab Bamrara

This Thesis has been read and approved by the examing committee:

Advisor: Dr. Hongning Wang

Advisor:

Committee Member: Dr. Dave Evans (Chair)

Committee Member: Dr. Haifeng Xu

Committee Member:

Committee Member:

Committee Member:

Committee Member:

Accepted for the School of Engineering and Applied Science:

Craig H. Benson, School of Engineering and Applied Science

May 2021

Acknowledgements

The past two years have been an incredible journey for me. I will never forget the joy and excitement of receiving an admit for my Masters' from the University of Virginia. Here, I ventured on different adventures and accumulated most of my research knowledge. Doing my thesis has been an wonderful experience for me, primarily because of the thrill of getting to know the unexplored areas. All of this would not have been possible without the help of the people I meet along the way in the form of Professors, Advisor, and friends.

First and foremost, I would like to express my sincere gratitude to my advisor Prof. Hongning Wang for giving me the opportunity to do research and providing invaluable guidance throughout this research. I thank him for his continuous support, patience, motivation, enthusiasm, and immense knowledge. He has taught me the methodology to carry out research and present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance. I could not have imagined having a better advisor and mentor for my Masters' thesis. Besides my advisor, I would like to thank the rest of my thesis committee: Prof. Dave Evans and Prof. Haifeng Xu, for devoting their time to provide me with encouragement, insightful comments, and challenging questions, which gave me a better understanding of my research.

I would also like to thank my friends who have always supported me throughout the journey. I want to specifically thank Huazheng Wang for providing me deep insights into my research work. I also thank Gaurav K. Jindal, Omkar Bhat, Arijit Pande, and Mukundan R. Mohan for giving me immense support and care amidst these challenging times.

I want to thank my dearest family - my mother and father who had given me all the support when I decided to come to UVA for my study, for all their wishes for my well-being, and their continuous emotional and spiritual guidance. Finally, I would like to thank God, the almighty, for his showers of blessings throughout my research work.

This work is partially supported by National Science Foundation under award IIS-1553568.

Abstract

Online Learning to Rank (OL2R) methods have been popularly studied in Information Retrieval (IR). The key idea is to estimate a ranker directly based on the interaction with users. Dueling Bandit Gradient Descent (DBGD) [1] is one of the most popularly studied OL2R methods, which is deeply rooted in online convex optimization [2]. However, little is known about the robustness of such algorithms in scenarios where a malicious attacker tries to change the user's feedback in such a way that the final learned ranker deviates away from what the user intended it to be. This work describes the real-world setting(s) on how a malicious attacker can use the characteristics of DBGD-based algorithm to attack it. We have also described in detail the threat model that we have used. We then propose two novel attacking strategies, namely, Naive intersection attack and Frequency attack, that the attacker can use on DBGD to divert the ranker towards the attacker's goal. The attacks and the corresponding deviation have been subsequently analyzed to show the robustness of DBGD based methods. The results show that DBGD is not robust against the proposed attacks. To the best of our knowledge, we are the first to propose poisoning attacks on an OL2R algorithm, show empirical results and analyze them.

Contents

1 Introduction

Ranking is one of the cornerstones of information retrieval (IR) to satisfy users' information needs. Today the web contains more than 5 billion web pages . We are overwhelmed by this flood of information from social media, news, and video websites. There is a need for a platform that can satisfy as much of our information need as possible in our limited time. Learning to Rank (LETOR) is an application of Machine Learning (ML) to create such platforms for IR. Given a list of documents, LETOR tries to find the appropriate ranking concerning a given query. Typically supervised learning approaches [3] are used for this purpose. In particular, for every query-document pair, there is a specified label attached, which suggests the relevance of the document for that particular query. These query-document pairs are usually represented as feature vectors. Thus, the goal of LETOR is to find the function that maps these features to the relevance labels. The features can be query-specific (Number of words in query), document specific (Page Rank, Document length), or can depend on both query and document (TF-IDF, BM25).

Traditionally, LETOR was done using annotated data in an offline fashion. Ranking a list of items was based on the implicit notation of relevance in each item. However, such notation was generally hard-coded by IR practitioners to create the annotated data. Machine Learning techniques were then used to find the mapping function. Although helpful, this method had various limitations. Firstly, the training data was limited as it was expensive to hire Professionals to annotate the data manually. Secondly, since there were multiple annotators, there was a chance that an annotator's preference might not align with other annotators; in such a case, it was challenging to select which annotation to use. Thirdly, there is a gap in what the annotator annotates and what the user wants [4]. Fourthly, when the dataset contain some personal information it can have some privacy concerns [5]. Moreover, once annotated, the data was fixed and cannot handle dynamic changes in the relevance labels [6].

https://www.worldwidewebsize.com/: Accessed on April 1, 2021

To mitigate these issues, OL2R methods are proposed [1, 7, 8]. The key idea is to estimate the mapping function directly based on the interaction with users. The interaction here refers to the feedback (i.e., clicks) that the users produce when presented with a document list. Hence, the ranker learns on the fly while interacting with the users. This solved some of the major drawbacks of the offline learning method. Firstly, data of users interacting with a search engine are often readily available [7]. Secondly, no annotators are required, and the ranker can also learn the changing relevance of the users directly from the feedback. Thirdly, there is no need to store sensitive data offline which alleviates many security issues.

Among many common approaches to OL2R, DBGD and its successor Multileave Gradient Descent (MGD) based algorithms have been vastly studied in the research community. Under DBGD based algorithms, an exploratory direction is proposed, which acts as a tentative model update direction. The ranking utility is then measured between the current direction and the exploratory direction. If the exploratory direction provides a better utility than the current direction, then the model is updated in that direction. The utility here is inferred from the feedback that the users give when presented with the interleaved list of documents from each ranker [9, 10]. In general, all the OL2R based methods depend on the user feedback (clicks), which helps model learning. The most common use of such approaches is in Search Engine Optimisation (SEO), where these methods are used to produce a ranking of documents. Recommendation systems can also use such a framework to learn implicitly from the user feedback [11].

The variety of benefits provided by OL2R approaches gives incentives to the attackers to influence their learning. For example, an attacker can promote a low-quality or untrustworthy document or promote its malicious website by influencing a particular algorithm. As the above approaches are not designed to work in adversarial environments, an attacker can compromise the learning system's confidentiality or integrity. In confidentiality attacks, the attacker tries to obtain information about the model being trained. In integrity attacks, the attacker's goal is to disrupt the algorithm's learning by modifying or inserting malicious data in the model training process. As such, integrity attacks provide more benefits to the attacker than confidential attacks [12].

The poisoning attacks are among the most common ways to make integrity attacks [13]. Here an adversary can inject malicious data into the training process such that the resulting system malfunctions as desired by the attacker [14]. The attack, as mentioned earlier, can be either a targeted attack, in which the attacker tries to make the system behave in a particular way or, it can be an indiscriminate attack where the attacker just wants to hamper the models' learning process. Based on the use cases of OL2R approaches, targeted attack gives more value to the attacker than the indiscriminate attacks. As explained previously, DBGD and MGD based algorithms depend on the users' clicks. However, there is no attempt to determine if the click received comes from the user or not. The adversary (aka. attacker) can use such a flaw to inject malicious clicks instead of regular user clicks to make the model deviate from what the user intended it to be.

The flaw mentioned above motivated our research to analyze DBGD and MGD based algorithms' robustness in the presence of an adversary. This work explains how the attacker can leverage the shortcomings of the algorithm to make targeted poisoning attacks. We also propose two novel targeted-poisoning attacks, namely, Naive intersection attack and Frequency attack and show the empirical results demonstrating the effectiveness of our proposed attacks. A thorough analysis of the performance of the attacks under different conditions is also presented. Our result suggests that DBGD and MGD based algorithms are not robust under the proposed attacks. While there has been promising research in adversarial attacks on ML [15, 16, 17] and supervised learning [18, 19, 20], little is known about adversarial attacks in OL2R methods. Hence, this work introduces a new paradigm of research in the Information Retrieval domain.

2 Related Work

ML algorithms are known to be vulnerable under adversarial examples [15]. An adversary can target the algorithm either by tampering with the training data or tampering with the model's inputs. Goodfellow et al. [16] have shown that the vulnerability of Neural Networks to adversarial attacks is in their linear nature. Following their work, Huang et al. [15] have proposed that the adversarial attacks effectively target Neural Network policies in reinforcement learning. They have proved this by showing how the existing crafting techniques for adversarial examples can also degrade the performance of trained policies. Lin et al. [17] have proposed methods to craft the adversarial examples and apply them in situations when Deep Neural Networks are training reinforcement learning agents. Two tactics are proposed in work, namely, strategically timed attack and enchanting attack. In the strategically timed attack, the authors made sure that the agent is getting attacked only in a small subset of time, which helps prevent detection. In the enchanting attack, the agent is lured to perform some sequence of actions. The closest research to our work has been with the Deep Ranking, in which the data samples are embedded into common embedding space, and then a distance metric is used to compute the ranking. Several works have used this workflow, including Deep-Image-Ranking [21] where the authors have created a multi-scale Neural Network to perform Image search, and [22] have used the workflow in cross-modal retrieval. However, only a little research is done in adversarial attacks on Deep Ranking. Li et al. [23] were the first to propose adversarial attacks using universal perturbation on image retrieval systems such that irrelevant results are returned to the user, while Liu et al. [24] have done the same by proposing adversarial queries.

Poisoning attacks, in particular, have also been studied extensively in the Supervised Learning domain. Biggio et al. [18] have proposed a poisoning attack on Support Vector Machines (SVM) using gradient ascent strategy. The motivation behind their work was the fact that many of the learning algorithms assume that the data being fed to them is well-behaved and natural. However, such a case is not always valid, and they have shown how an intelligent adversary can use malicious data to attack the algorithm. Mei et al. [20] have

proposed an optimal training set attack and have shown that such attack can be formulated as a bilevel optimization problem. Further, they show that learners with certain Karush-Kuhn-Tucker conditions can efficiently solve such problem using gradient methods. Using the strategy, an attacker can contaminate the training set such that the trained model is beneficial for the attacker, and the authors have demonstrated these attacks on SVM, Logistic Regression, and Linear Regression. Xiao et al. [19] have investigated the robustness of feature selection methods and have shown that these methods can be significantly compromised under malware detection during adversarial attacks. The authors have analyzed a real-world security application using PDF malware detection. They have shown how an adversary can control the methods by injecting only a tiny fraction of poisoned points. Liu et al. [25] have proposed a general framework for data poisoning attacks on the graph-based Semi-supervised learning approaches. Zhang et al. [26] have studied poisoning attacks on the online setting where the adversary can contaminate the data stream. Here, the authors have framed the online poisoning attacks as a stochastic optimal control problem and have proposed two attacking strategies. Both the strategies are shown to perform near-optimal attacks on both supervised and unsupervised learning tasks.

3 Background

3.1 Dueling Bandit Gradient Descent and Multileave Gradient Descent

DBGD is one of the most commonly used approaches for OL2R problems based on vector space model [27]. As mentioned before, the query-document pairs are represented as feature vectors, and the goal of a L2R algorithm is to learn a function that maps these features to relevance labels. In a vector space model, the function to learn is represented by a weight vector in an n-dimensional space (n being the dimension of the feature vector). Given an initial weight vector (w_0), the algorithm tries to find the optimal weight vector (w^*), which signifies each feature's importance to the ranking problem. In every iteration of DBGD, the algorithm proposes a new exploratory vector ($w_t^{'}$) by perturbing the current weight vector (w_t) by a random unit vector (u_t). The two vectors propose two different rankings of documents, and then an interleaved ranking is presented to the user. Based on the users' feedback, the two vectors are then compared, and if the exploratory vector wins, the current weight vector is updated. The equations to propose an exploratory vector (Eq. 1) and perform updates (Eq. 2) are represented. Here δ and γ are constants representing exploration and exploitation step-sizes, respectively. Figure 1 shows the overview of the DBGD algorithm.

1. $w_t^{'} = \mathbf{P}_W(w_t + \delta u_t)$; `// projected back into W`

2. $w_{t+1} = \mathbf{P}_W(w_t + \gamma u_t)$; `// also projected`

MGD is a generalized version of DBGD, where the algorithm can have multiple exploratory rankers as opposed to a single exploratory ranker used in DBGD. The working remains the same. At every iteration, the algorithm proposes multiple exploratory vectors by perturbing the current weight vector (w_t). A multileaved ranking of all the vectors is then presented to the user. Based on the feedback, the ranker gets updated either based

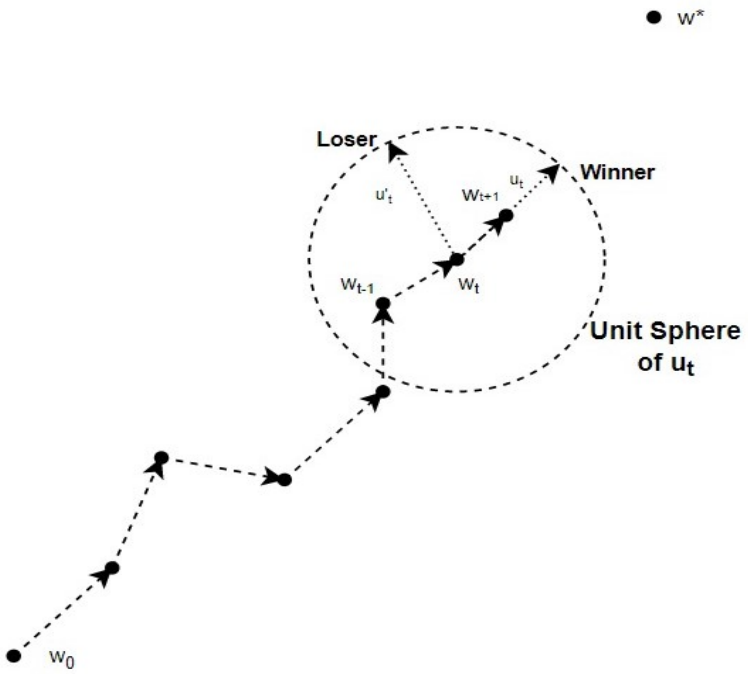


Figure 1: Overview of DBGD algorithm

on the mean of all rankers who won (MGD - mean winner) or only one of them (MGD - winner takes all). Doing this variation allows MGD to better select candidates and, hence, learn faster than DBGD [7]. The critical insight in DBGD and MGD based algorithms is to estimate gradients in an unbiased way using random sampling. Since the gradients are chosen from the user feedback, there is an opportunity for the attacker to inject some negative feedback such that the gradients are sent in a different direction. Putting this in the above context, the attacker can have its weight vector (w_a), which is different from the optimal weight vector that the user is trying to converge to (w^*). The goal of the attacker is to make the model converge to the attackers' specified direction. Figure 2 shows the overview of DBGD when the attacker is present.

3.2 Interleaving and Multileaving

Interleaving is an evaluation technique used to compare rankers by combining their ranked lists into a single list and then tracking the user clicks. Based on these clicks, the winner is determined. Several variations exist on creating an interleaved list and assigning

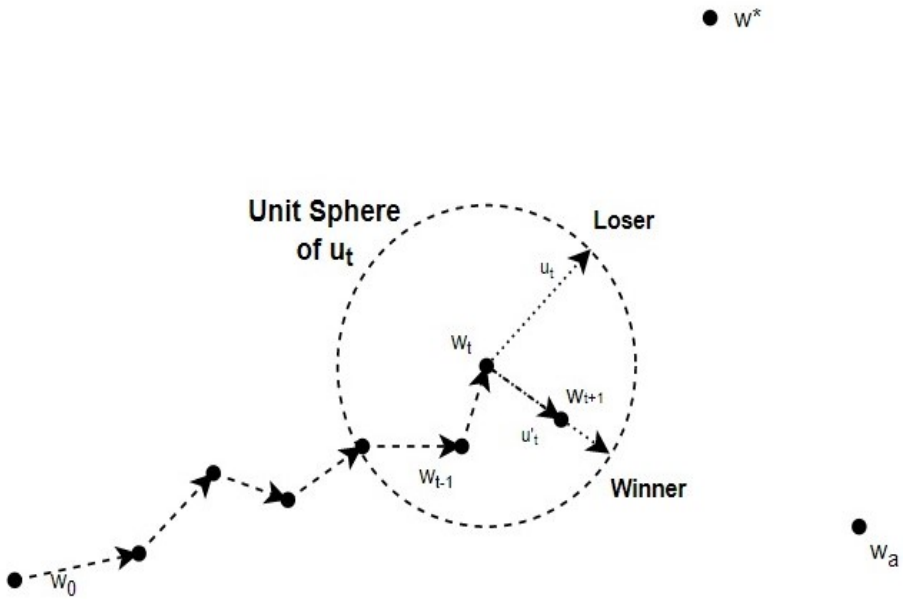


Figure 2: Overview of DBGD in the presence of an attacker

scores for each of the clicked documents. Team Draft Interleaving (TDI) [10] is one of the most common ways for creating an interleaved list. This technique simulates a process of two captains picking their team members. Say we have R1 and R2 to be the preference lists of two captains. At every turn, a coin is tossed to determine which of the captains will pick first. While picking, the captains select their preferred team member from their preference list (R1/R2) who has not been picked up yet. This process continues until all the members are picked. Additionally, the information about which captain picked the team member is stored to assign credits after winning. Figure 3 shows an overview of the Team Draft Interleaving. When more than one rankers are compared, Team Draft Multileaving (TDM) [28] method is used. TDM uses the same methodology, however; now, multiple captains take a turn while selecting members.

3.3 Learning Rate

Learning Rate is the most crucial hyper-parameter in ML algorithms [29]. It determines the step size at each iteration while moving toward a minimum of a loss function [30]. While fixing a learning rate for an algorithm, there is always a trade-off between the rate of

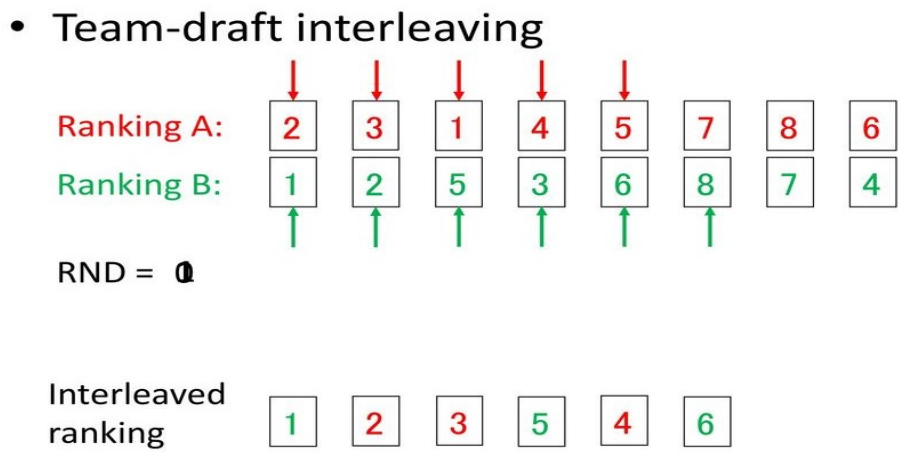


Figure 3: Team Draft Interleaving

convergence and overshooting. If the learning rate is set to a high value, this results in the loss function's divergent behavior(oscillation). However, if it is set to a low value, then the model is prolonged to converge. Figure 4 shows the two scenarios. As such, it is tricky to set up a fixed learning rate for the entire training process. To deal with this issue, Learning Rate decay (lrDecay) is used as a *de facto* technique for training modern ML algorithms. Here initially, a high learning rate is set, which suppresses the memorization of noisy data, and later the learning rate is decayed to make sure that the model learns complex patterns [31]. In this work, we show the effects of Learning rate and lrDecay on the learning process of DBGD and MGD based algorithms. We further describe how the attacker can use the learning rate information to perform data poisoning attacks.

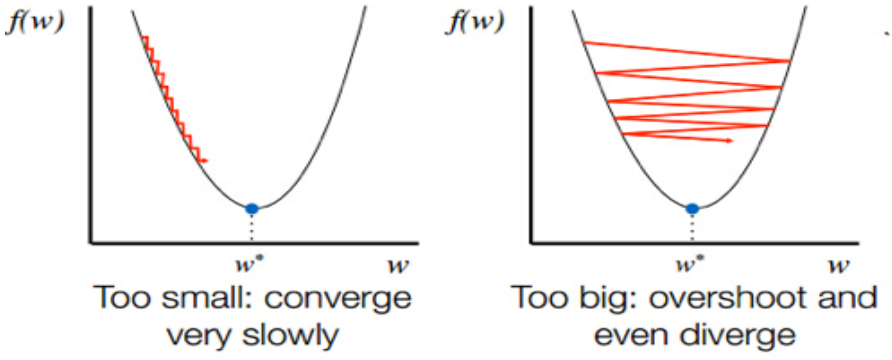


Figure 4: Effect of Learning Rate

CS-6501 Information Retrieval, UVA
https://www.upgrad.com/blog/gradient-descent-algorithm

4 Methodology

This section firstly presents our Threat model in which we have worked. Then we describe the two different types of attack we propose and the real-world scenario on how these attacks can be performed. A user is a person with basic network access via web browser. An IR system is a web server offering services using a database of documents. The general workflow of user interaction with an IR system is shown in Figure 5. It works in five stages. Firstly, the user has some information need for which he queries the IR system. The IR system generates a ranked list of documents for the query and then returns it to the user. The user provides the feedback/clicks to the IR system, based on which the ranking model's learning happens. As discussed, feedback is essential for the learning process. If the attacker somehow alters this feedback, then the ranker should not learn what the user wants.

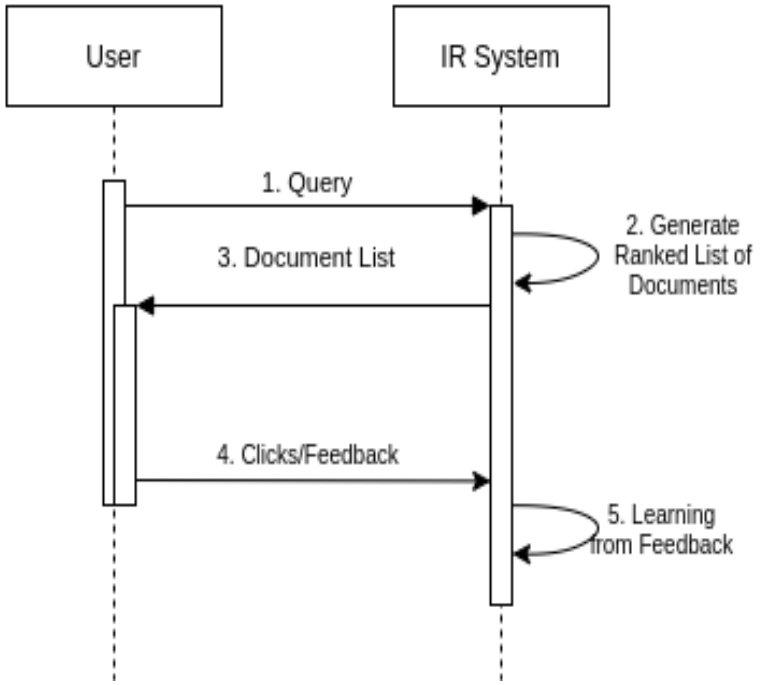


Figure 5: Normal workflow of User Interaction

4.1 Threat Model

4.1.1 Attacker's Goal

We consider that the attacker wants to promote a particular set of documents which are governed by attacker's weight vector (w_a). Specifically, if the IR system returns a ranked list of documents to the user, the attacker wants its documents to be present in the returned list. Sometimes, the attacker just wants to mislead the learner, in such cases it can choose a random weight vector ($w_a^{'}$) and then produce its feedback. Moreover, if the attacker wants to demote some documents, it can do so by promoting other documents [32].

4.1.2 Attacker's Capabilities

While designing the attack, the first challenge was to decide on the type of attack that is feasible in the above scenario. The attacker's final goal is to change the feedback given to the IR system while making sure that the system assumes that the feedback is coming from the user. Based on the requirements, we propose that the attack should be similar to the man-in-the-middle attack. In this type of attack, the attacker takes control of the ongoing network communication between the two parties (user and IR system) [33]. This makes it possible for the attacker to eavesdrop and even change the communication between the user and the IR system. If done precisely, neither of the end systems is aware that the communication is compromised, and they continue to share information. In this work, we have specifically assumed that the attacker can change the information being sent from the user to the IR system. Additionally, the attacker can eavesdrop on the data being sent in both directions.

4.1.3 Attacker's Background Knowledge

Our second challenge was to decide on the data available to the attacker. This is crucial because if the attacker has to change the feedback, it needs to know how to do so. In other

words, the attacker needs to produce its feedback by looking at the list of documents. It should be able to make a decision on which documents to click and which ones not to. After careful consideration, in this work, we have assumed that the attacker knows about the feature vectors being used by the model. If the attacker knows which feature vectors are being used (ex. BM25, TF-IDF), then it can compute those features for the documents returned by the IR system based on the query the user proposed. If the attacker has such critical information about the model, it can construct its weight vector (w_a). The attacker can then later use (w_a) in an offline manner to find its preference list for every query that the user asks. If the attack is successfully completed, then after learning, the document list returned by the IR system should be similar to the attackers' preference list. This notion of similarity is captured by the evaluation metric we have used in the experiment section.

4.1.4 Attacker's Limitations

In a real world scenario, it would be hard for the attacker to change every feedback of the user. The attacker also will not want to take a risk to being detected if it tries to attack every interaction of the user. Keeping this in mind, we have assumed that the attacker is limited by the number of queries/interactions it can attack. The attacker's goal now becomes to maximize it's effect under these limitations.

4.2 Real-World Scenario

In the real world, any SEO person can become an attacker. The typical working of SEO makes sure that the relevant results are being returned to the user, and thus it tries to promote specific documents to the user. If the Search Engine itself wants to promote a particular agenda, then it will make sure that the returned results also reflect that agenda. If the Search Engine is benevolent, an external browser application or extension can work as an attacker. Sometimes the browser might get jeopardized; in such case, the attacker can control the direct communication of the browser with the server. In some other cases, an

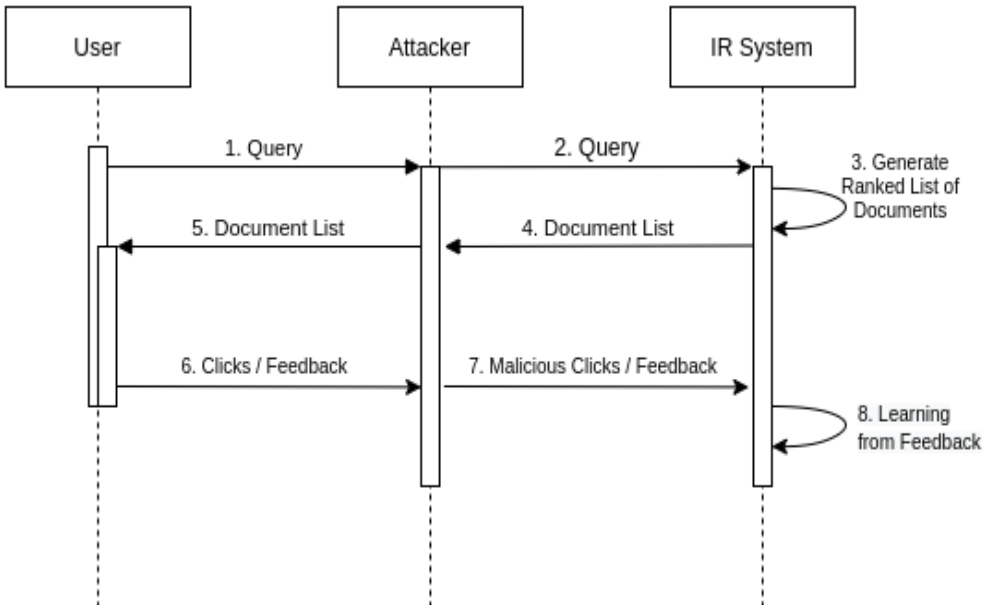


Figure 6: Malicious workflow of User Interaction

external browser extension can also act maliciously in the background. Since the internal working of the extensions such as Ad-blockers is unknown, it might sometimes happen that the blocker can even block some of the relevant data for the user. Other times the extension can also generate its feedback on top of the user's feedback. Figure 6 explains such a real-world scenario on how an attacker/application can intercept the user query and feedback to inject erroneous data in the learning process.

As before, the user has some information need, and it queries the IR system. The attacker can intercept the query to know the details and forward it to the IR system. The IR system then sends the document list back, which is again intercepted by the attacker and then forwarded to the user. The user provides the feedback, which the attacker rejects and sends its malicious feedback to the IR system. Since the model has no discrimination for the feedback, it will learn from whatever feedback it receives. Using the mentioned technique, the attacker can perform a data poisoning attack on the IR system. In this work, we have presented two novel attacking strategies that the attacker can use to send malicious feedback to the IR system. The following section explains these strategies.

4.3 Naive intersection attack

We have assumed that the attacker can intercept the communication between the user and the IR system. As such, the attacker knows about the query being asked by the user and also the document list being returned by the IR system (Step 1 and 4 in Fig. 6). Based on the data available with the attacker, it can have a mapping function (w_a), using which the attacker can find out its own ranking of the documents which are present in the returned document list. Let us call this ranking an Attacker Ranking (AR). The final goal of the attacker is to make the ranker learn its mapping function. The ranker also has its mapping function (w_t) that it has learned based on previous interactions with the user. Ranker will also produce a ranking of documents based on its function. Let us call this ranking a Train Ranking (TR) as the ranker is still in the learning phase.

Given TR and AR, the attacker can then stop the regular user clicks from going through and generate its clicks based on the two rankings' intersection. Algorithm 1 describes the way the attacker can change the user feedback. Here we have two hyper-parameters, start, and end, which specifies the positions that the attacker is most interested in. For a specific position, say the first position in the AR, the attacker can set start = 0 and end = 0 to ensure that the top-most document in the AR is clicked in the TR. If the end is set to 2, then the attacker will make sure that its top 3 documents are clicked in the TR. This attack is termed as intersection attack because the attacker tries to find a sub-list governed by start and end, and then intersect it with the TR. Figure x shows an example where start and end are set to 0 and 2 respectively.

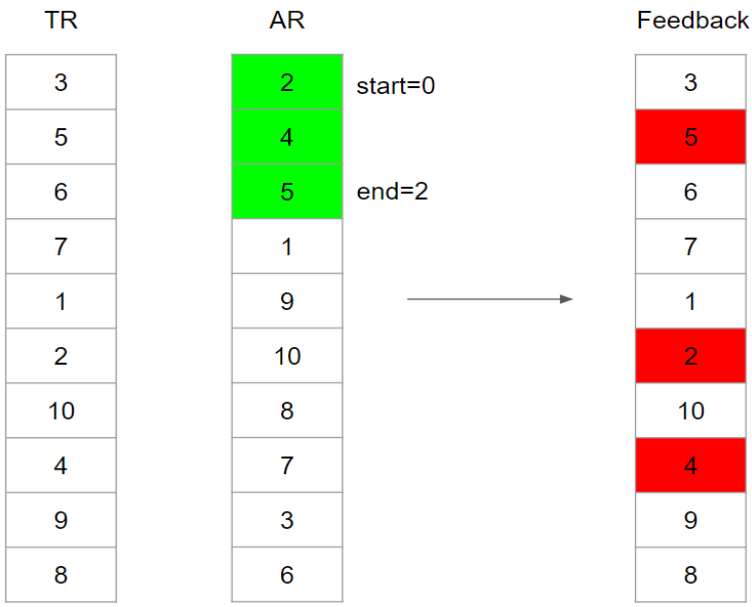


Figure 7: Naive Intersection attack example

Algorithm 1: Naive intersection attack

Result: Malicious clicks

Input: TR, AR, start, end

clicks ← [ ];

for $i \leftarrow 0$ **to** $length(TR)$ **do**

  if *TR[i] in AR[start:end]* **then**

    clicks ← clicks + True;

  else

    clicks ← clicks + False;

  end

end

4.4 Frequency attack

Previous attacking strategy is termed as Naive because the attacker assumes the TR to be a proxy of the actual model's ranking. As such no further attempts are made to check if the clicks are meaningful. One of the notable characteristics of DBGD based algorithms is the use of an exploratory ranker while creating the TR. As explained previously, the TR is not the model's actual ranking; instead, it is the interleaved ranking that the model proposed. Hence, TR should not be considered as the ground truth for the model's current state. Instead, the attacker needs to find out which TR documents come from the current ranker and which documents come from the exploratory ranker. As we have discussed, knowing such information is important because the model updates can only happen if the exploratory direction wins; hence, while generating clicks, the attacker also needs to make sure that it is clicking the document which is proposed by the exploratory ranker.

Our next challenge was finding a way to provide the attacker with such information without giving it an unfair advantage. For doing this, we visit the DBGD algorithm once again. The algorithm looks pretty intuitive; however, there is a minor flaw in the algorithm. What happens when the user does not provide any feedback? Going through the update mechanism, the model will not update in that scenario and remains the same. In technical terms, the weight vector (w_t) remains at the fixed location. This is important because we do not want updates to happen in the wrong direction when the user does not provide any feedback. The attacker can use this property of the DBGD algorithm to know the required information about the documents.

In particular, let us assume that the attacker can also query the IR system on the user's behalf. This assumption also depends on the nature of the man-in-the-middle attack. If the communication is highly compromised, it is easy for the attacker to send in queries on behalf of the user. In such conditions, the attacker can then repeatedly ask the same query to the system. Every time the system will produce a different interleaved ranking as the exploratory direction ($w_t^{'}$) will be different. However, the ranking produced by the current rankers' direction (w_t) will be consistent. The attacker can create a frequency table (freq)

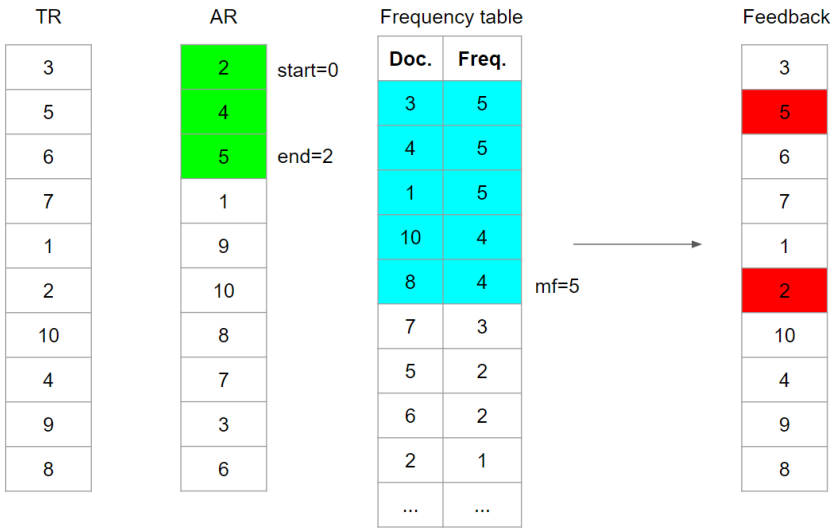


Figure 8: Frequency attack example

of the documents being presented every time. The key idea here is that the frequency of the documents which are given by the current ranker will be higher, as compared to the documents given by the exploratory directions. This is because since the ranker is fixed, it will always promote a specific ranking, while the exploratory ranker will not.

Algorithm 2 describes how the attacker can generate clicks using this technique. Here we have three hyper-parameters, two of which are the same as the previous attack (start, end). The third one (mf) represents the number of most frequent documents the attacker believes are from the ranker. From this mf, the attacker will create a top-k list, which acts as a proxy of the current ranker's ranking. While doing the intersection, attacker will make sure that none of the documents from this top-k list are clicked. Figure x gives an example where start, end and mf are set to 0, 2 and 5 respectively.

4.4.1 Deciding a Stopping Criterion for Frequency Attack

For defining the stopping criterion, either of the two things can be done:

1. Fix a particular number to denote how many times the attacker should repeat the same query.

2. Make a dynamic decision based on the distribution of the frequency table.

Algorithm 2: Frequency attack

```
Result: Malicious clicks
Input: TR, AR, freq, mf, start, end
clicks ← [ ], top_k ← [ ], i ← 0;
sorted_freqs = sort(freq, desc) ;          // Sorting freq in descending order
while i < length(sorted_freqs) and i < mf do
    top_k ← top_k + sorted_freqs[i];
    i ← i + 1
end
for i ← 0 to length(TR) do
    if TR[i] not in top_k and TR[i] in AR[start:end] then
        clicks ← clicks + True;
    else
        clicks ← clicks + False;
    end
end
```

Fixing a number is easy, and it does provide the required benefits. However, sometimes the attacker might require more repeats to get enough information, while at the other times, a small number of repeats might do the trick. As such a dynamic stopping criteria for the attacker is beneficial. Our next challenge was defining such a criterion based on the information the attacker already has. For this purpose, we looked at the frequency distribution of the documents in the frequency table. We ran an experiment for the frequency attack by fixing the number of repeats to be nine on the MQ2007 dataset (explained in the later section). Table 1 shows the distribution that we see initially, at the 500th iteration and then after the 1000th iteration. These iterations are chosen to see how the distribution changes with time.

By looking at the Table 1, we came up with an understanding of the distribution. Firstly, at the 0th iteration, since all the model weights are initialized to a 0 value, it cannot present any preferred ranking. For this reason, we can see a near to flat distribution even in the later repeats of the 0th iteration. However, as we move towards the later iterations, the model weights have become non-zero, so it can now propose a separate ranking list. As can be

seen from the table, the distribution is not flat in the later repeats of the 500th and 1000th iteration. Secondly, we also noticed from the above table that as the standard deviation (sd) reaches a value greater than 1, the frequency of some of the top results always increases with the repeats. For example, let us look at the 4th repeat for the 500th iteration. The first five documents have a frequency of 4, and with the later repeats, this frequency increases by one every time. This means that these documents are being presented in every repeat of the same query. As we know from the discussion, that only the current ranker with a fixed weight vector w_a can be consistent with the ranking. Hence we can assume these documents to come from the current ranker and not from the exploratory ranker.

Table 1: **Distribution of the frequency table**

Repeat	0^{th} **iteration: sd**	500^{th} **iteration: sd**	1000^{th} **iteration: sd**
1	[1,1,1,1,1,1,1,1,1,1]: 0.0	[1,1,1,1,1,1,1,1,1,1]: 0.0	[1,1,1,1,1,1,1,1,1,1]: 0.0
2	[2,2,1,1,1,1,1,1,1,1]: 0.4	[2,2,2,2,2,2,2,1,1,1]: 0.45	[2,2,2,2,2,1,1,1,1,1]: 0.5
3	[2,2,2,2,2,2,1,1,1,1]: 0.49	[3,3,3,3,3,3,2,2,1,1]: 0.8	[3,3,3,3,3,2,2,1,1,1]: 0.87
4	[3,3,2,2,2,2,2,2,2,2]: 0.39	[4,4,4,4,4,4,2,2,1,1]: 1.26	[4,4,4,4,4,3,2,2,2,1]: 1.09
5	[3,3,3,3,2,2,2,2,2,2]: 0.49	[5,5,5,5,5,4,3,2,2,2]: 1.32	[5,5,5,5,5,3,3,2,2,2]: 1.34
6	[4,4,3,3,3,3,2,2,2,2]: 0.75	[6,6,6,6,6,4,4,2,2,2]: 1.74	[6,6,6,6,6,3,3,3,2,2]: 1.73
7	[5,4,3,3,3,3,3,3,3,3]: 0.64	[7,7,7,7,7,5,5,2,2,2]: 2.16	[7,7,7,7,7,4,3,3,3,3]: 1.92
8	[5,5,4,4,3,3,3,3,3,3]: 0.8	[8,8,8,8,8,6,5,3,3,2]: 2.34	[8,8,8,8,8,4,4,4,3,3]: 2.23
9	[5,5,5,4,4,3,3,3,3,3]: 0.87	[9,9,9,9,9,7,5,3,3,3]: 2.65	[9,9,9,9,9,5,4,4,3,3]: 2.65

This can help define a stopping condition. The next challenge then arises on how the attacker can determine the documents that come from the current ranker. For this issue, we introduced a new hyper-parameter **sd_constant**. The attacker's goal now becomes to look for a position in the above distribution which is at least **sd_constant** standard deviations away from the next position. This is done to check if there are any abrupt changes in the frequency. This position can later be used to override the value of **mf** in the Frequency attack. Thus the **sd_constant** controls this abrupt change. Tuning this

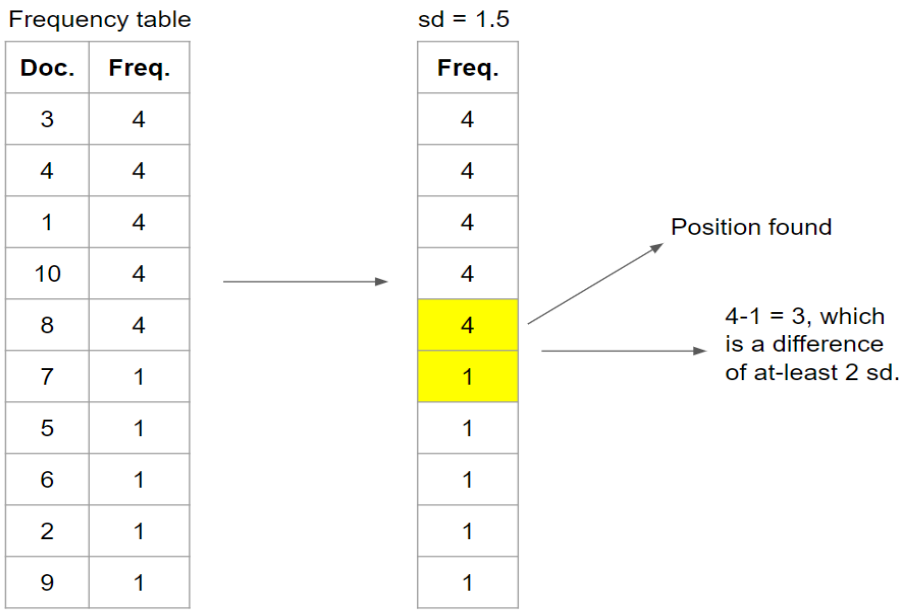


Figure 9: Finding a stopping position

hyper-parameter can be a bit tricky; however, we can use some of its characteristics. If the value of **sd_constant** is set too large (say, 3.5 or 4.5), then a lot more repeats will be required to find the position. If it is set too low (say close to 1), then the repeats will stop very quickly, but the attacker will not get enough information about the current ranker. As this research aimed to discuss the possibility of attacks, we have not carefully fine-tuned this hyper-parameter and have fixed the value of **sd_constant** to be 2.0 to tackle both the extremes. Moreover, if we assume the frequency to follow a normal distribution, then almost 95% of the data will lie within the 2.0 standard deviations. While implementing the algorithm we made use of both the stopping criterion. Atmost 9 repeats to the same query will be there. If before 9 repeats we found the position of the abrupt change, then the attacker can stop further repeats of the same query. Figure x shows such a scenario where the attacker can stop going for further repeats. The position can be used to create a proxy of the current ranker's ranking, which can be given to the Frequency attack algorithm. Algorithm 3 describes how the attacker can make such a dynamic decision by looking at the frequency distribution.

Algorithm 3: Dynamic Stopping criteria for Frequency attack

```
Result: Frequency table, index
Input: query_id, ranker, sd_const
freq ← { }, index ← 0, repeats ← 0;
while repeats < 10 do
    TR = ranker.get_ranking(query_id) ;    // Getting the interleaved ranking
    for r ← TR do
        if r in freq then
            freq[r] ← freq[r] + 1;
        else
            freq[r] ← 1;
        end
    end
    sorted_freqs = sort(freq, desc) ;        // Sorting freq in descending order
    i ← 0 ;
    while i < length(sorted_freqs) and i < mf do
        top_k ← top_k + sorted_freqs[i];
        i ← i + 1
    end
    sd ← standard_deviation(top_k) ;  // Get the standard deviation of top_k
    if sd > 1 then
        found ← False;
        for ind ← 0 to length(top_k) − 1 do
            if sd_const*sd ≤ top_k[ind] - top_k[ind+1] then
                found ← True ;                                  // Index found
                index ← ind;
                break;
            end
        end
        if found then
            break ;                              // Breaking from the while loop
        end
    end
    repeats ← repeats + 1
end
```

4.5 Picking an Attacking Strategy

Both of the algorithms have their pros and cons. Depending on them, an attacker may decide to launch one type of attack instead of the other. A naive intersection attack is preferred when the attacker wants to attack quickly. This type of attack is also relatively easy to implement and can be used in a real-world scenario pretty effortlessly. One of the drawbacks of the naive attack is that there are too many clicks that the attacker has to generate. This is not possible every time as the attacker might be bound by the number of clicks it can produce. In such cases, frequency attack shines, as while doing this type of attack, fewer and meaningful clicks are generated. However, it becomes tricky to implement, and it has more hyper-parameters to tune. Additionally, it requires that the attacker can query the IR system on behalf of the user. If we consider the number of queries that an attacker can attack as its budget, then a naive intersection attack can perform better than the frequency attack. However, if the attacker has a budget on the number of clicks it can generate, then a frequency attack would provide better updates than the naive attack.

5 Experiment Settings

5.1 Datasets

We have used 4 LETOR datasets [34] to show that our proposed attacks are feasible. These include MQ2007 [35], MSLR-WEB10K [35], Yahoo! Learning to Rank Challenge [36] and TD2003 dataset. In each of these, the query-document pair is encoded as a feature vector. Some of the features include PageRank, BM25, TF-IDF URL length, and many more depending on the dataset. Out of the three, TD2003 is the smallest dataset containing less than 150 queries, but with 1000 assessments in each one of them. Documents in this dataset are collected from the .GOV collection, which is crawled from the .gov domain, and binary relevance labels are used here. MQ2007 dataset contains 1700 queries, including informational and navigational queries. The feature vector is of 46 dimensions, and the relevance label is divided into three categories: 0 (non-relevant), 1 (relevant), and 2 (most relevant). The MSLR-WEB10K dataset was released by Microsoft also in 2010. It consists of 10,000 queries and 136-dimensional feature vectors. The relevance labels again range from 0 (not relevant) to 4 (most relevant). The Yahoo! Learning to Rank Challenge dataset was released in 2010 by Yahoo! to promote research in the learning to rank algorithms. This dataset contains around 36,000 queries, 883,000 assessed documents, and 700 ranking features. The relevance labels also range from 0 (not relevant) to 4 (most relevant).

5.2 Modelling User Behaviour

We have used a standard setup for this project provided here [37]. Cascade click model [38] is used for simulating the user clicks under this setup. This model assumes that the user scans the documents linearly in a top-down fashion and clicking the relevant document(s). The probability of clicking a document is conditioned on the relevance label of the document. Also, since the user moves in a top-down fashion, documents at the top might already fulfill the user's information need. So, after evaluating each document, the user decides to continue

examining the list further or stop. The probability distribution from this behavior is again conditioned on the relevance of the examined document. We can change these probabilities to simulate different user interactions. For this project, we have used expert probability configuration (exper_1) for simulating regular user clicks. Under this configuration, the user will click at-max one document which is the most relevant. This is done to give both the attacker and the user a similar advantage which will be clear in the later section.

5.3 Evaluation Metrics

OL2R algorithms' performance is generally measured in Normalized Discount Cumulative Gain (NDCG) [4]. NDCG is a metric for measuring the ranking quality based on the relevance labels. It can also be interpreted as the extent to which a given ranking of documents is in agreement with the ideal ranking of relevance labels. This ideal ranking can be computed by sorting the documents based on the relevance label scores. The primary advantage of NDCG is that it can use the graded relevance labels present in the dataset. Moreover, NDCG promotes rankers based on the position of the relevant documents. If one ranker puts relevant documents in a higher position than the other ranker, then its NDCG score will be higher than the latter. In summary, NDCG is consistent with the Cascade Click model's assumption, where the user scans the documents from top-to-bottom. The standard setup already provides the implementation of NDCG; however, we have created the relevance labels for the attacker. While doing so, we made use of binary relevance labels: 0 (not relevant), 1 (relevant). If the document is among the top-5 documents in the attackers' list, then it is relevant; otherwise, not. Two NDCG scores are presented in the performance graphs. NDCG attacker refers to the performance score when the attacker's labels are considered while generating the score. This means that NDCG attacker represents the agreement of the current model ranking with the attacker's ranking. NDCG label refers to the performance score when the ground truth labels (already present in the dataset) are considered while generating the score. NDCG label represents the agreement of the current model ranking with the ground-truth ranking. Thus, the attacker aims to make sure that

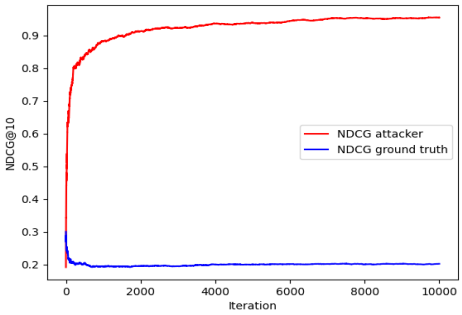
the NDCG label stays the same or decreases while its NDCG attacker increases. Since the weight vectors of the ground truth model (w^*) and that of the attacker (w_a) are different, the relevance labels concerning a query-document pair will also be different for them. We also show the learning rate curves for the models, which will help in explaining the results.

6 Results and Discussion

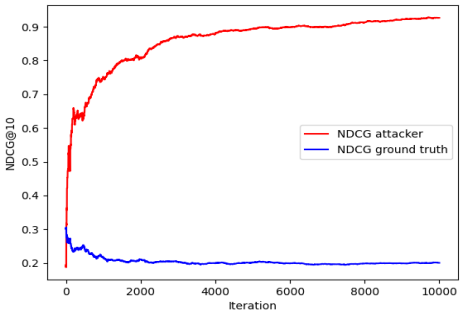
6.1 Attacks on DBGD

Firstly, we show the NDCG performance graphs for all the datasets, implying that our attacks are working (Fig. 10). We run our experiment for 10000 iterations. Each experiment is run ten times on each fold in the MQ2007, TD2003 and Yahoo dataset, while it was run five times for the MSLR dataset because of size and time limitation. Hence the numbers in the graph show the averaged performance results. The number of results being presented to the user is set to 10. For the naive intersection attack, start and end are set to 0 and 1 respectively. This is done to showcase the strength of the naive attack even when the attacker is only allowed to click one document per query. For the frequency attack, mf is set to 5, sd_const is set to 2.0, start and end are set to 0 and 5 respectively. This is a conservative approach where the attacker believes that top-5 frequent documents would have come from the current ranker. In all the three datasets, the NDCG with respect to the attacker increases until it saturates, and the NDCG concerning the ground truth labels also saturate but at a lower value. This means that the ranker can go in the direction as the attacker planned, which shows the attacking strategies' success.

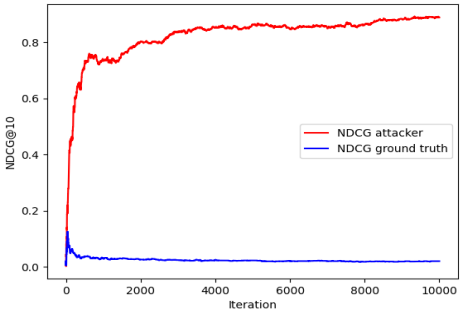
Looking at the results, we can also see that the Naive intersection attack is much faster than the Frequency attack. This was expected as frequency attack only works once for several queries while the naive attack works for every query. Although frequency attack is slow, we can see that it can indeed reach a similar performance as the naive attack in the later iterations. Another concerning problem in the above attacks was that the attacker had



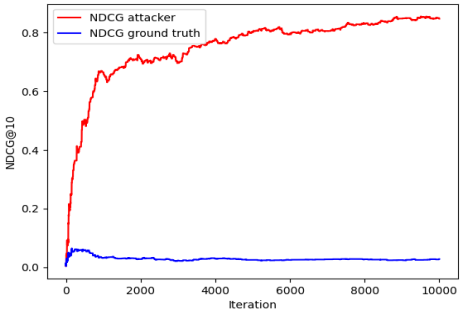
(a) Naive Intersection Attack MQ2007



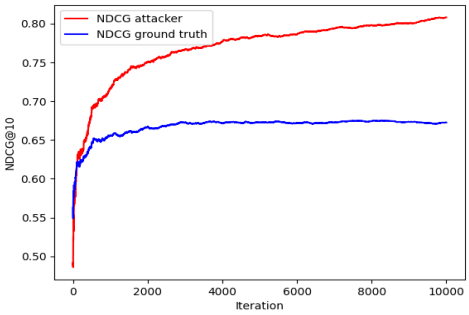
(b) Frequency attack MQ2007



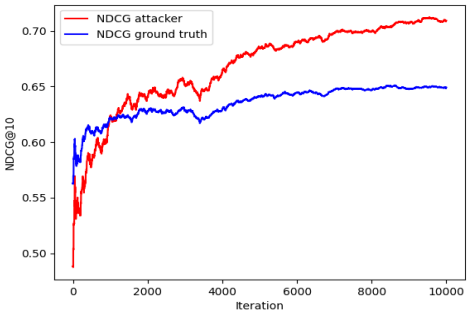
(c) Naive Intersection Attack TD2003



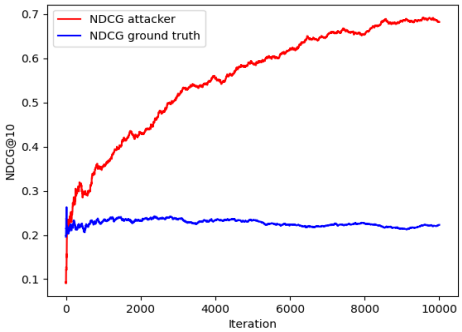
(d) Frequency attack TD2003



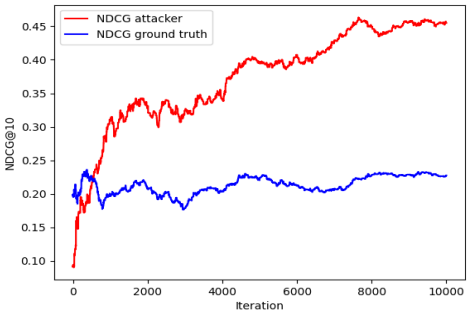
(e) Naive Intersection Attack Yahoo



(f) Frequency attack Yahoo



(g) Naive Intersection Attack MSLR

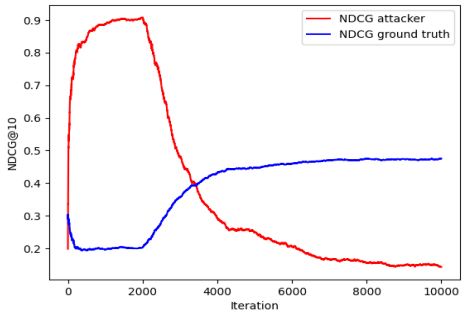


(h) Frequency attack MSLR

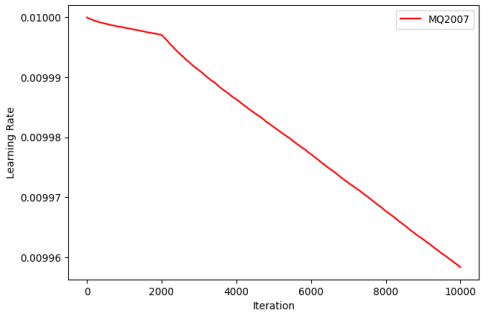
Figure 10: Effectiveness of attacks on DBGD

complete control over the users' clicks, which cannot be possible every time. The attacker generally has a budget for either the number of iterations (queries) that it can attack a user or the number of clicks it can generate, and many others. We wanted to see how the attacks would perform if the attacker were constrained just like in real world scenario. Considering this in mind, we gave the attacker a limited budget of 2000 iterations that it can attack out of 10000 iterations, which depicts a real-world scenario. We also focus our successive results on the Naive-intersection attack as it is fast, and the attacker can attack many more queries in this attack than the Frequency attack. Since the attacker only clicks at most one document, **exper1** click model is used for generating normal user clicks. This is done to ensure that the clicking behavior is similar in both the attacker and the regular user. Doing so also enables us to make apple-to-apple comparisons. Figure 11 shows the performance of the Naive intersection attack when the attacker has a limited budget. In the above results, the attacker attacks the starting 2000 iterations and the rest of the iterations the user is regularly clicking. From the performance figure, it can be seen that the NDCG score concerning the attacker increases for the first 2000 iterations in all the datasets, after which it quickly comes down. However, performance with respect to the ground truth was almost constant for the starting 2000 iterations, after which it rises to a saturation point. We concluded from the above results that DBGD under the default parameter settings could not hold the NDCG performance.

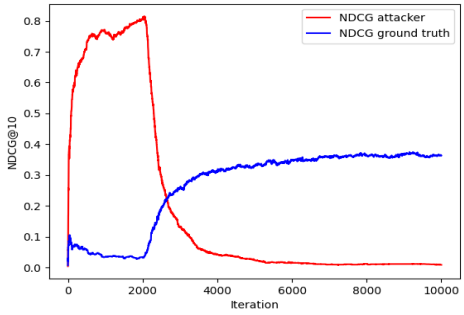
While analyzing the results, we noticed that the learning rate of the DBGD algorithm (Right side images on Fig. 11) had not decreased significantly from the initial settings when the attack finishes. We hypothesize that the non-changing learning rate might be why the DBGD is unable to hold the NDCG performance. If we can quickly reduce the learning rate while the attacks are happening, it might be possible for DBGD to hold the NDCG score after the attacks are finished. To quickly decrease the learning rate, we looked at the default implementation of the DBGD algorithm. We noticed that the learning rate decay by default was set to a value much closer to 1 (0.9999977) when going through the details. This means that the learning rate is almost constant as it changes very slowly with the updates. For proving our hypothesis, we decreased the learning rate decay to 0.99, and results for the



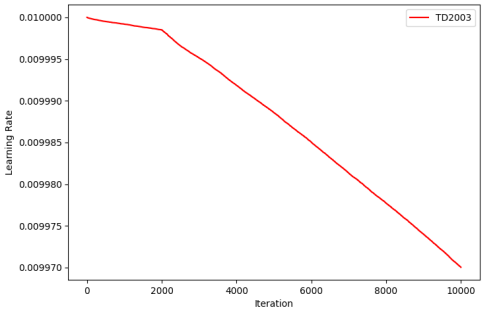
(a) Naive Intersection Attack MQ2007



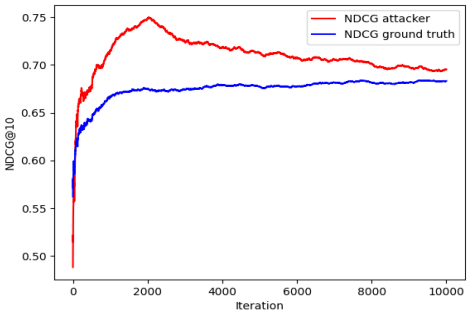
(b) Learning Rate MQ2007



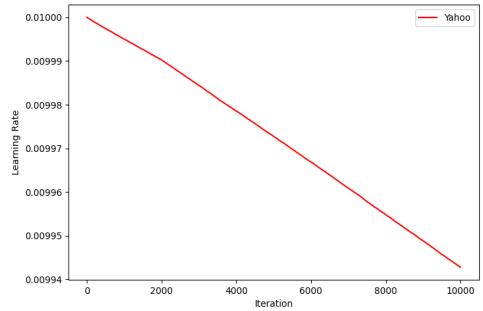
(c) Naive Intersection Attack TD2003



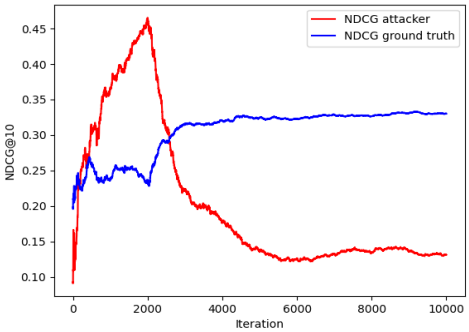
(d) Learning Rate TD2003



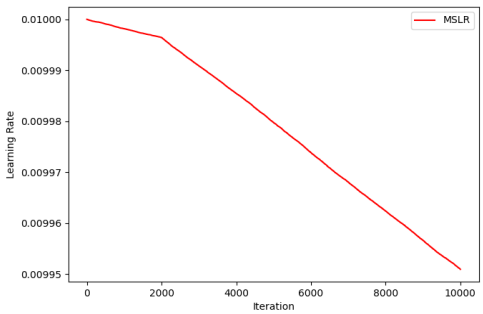
(e) Naive Intersection Attack Yahoo



(f) Learning Rate Yahoo



(g) Naive Intersection Attack MSLR

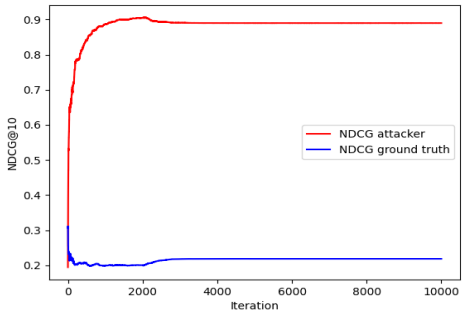


(h) Learning Rate MSLR

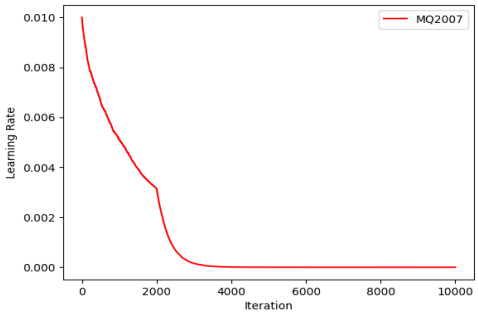
Figure 11: Limited Budget attack

same are shown in Figure 12. Concurrent with our hypothesis, for the MQ2007, Yahoo and MSLR datasets, the algorithm can hold the NDCG performance after the attack finishes. Additionally, we also show the learning rate graphs, and as can be seen, it quickly reaches a near to 0 value at the end of 2000 iterations for the two datasets. In the TD2003 dataset, the learning rate has not decreased that much drastically as compared to the other datasets. This is the reason why the performance in TD2003 decreases after 2000 iterations. However, after 4000 iterations, we can see that the learning rate for TD2003 has also decreased to a value closer to 0. After this point it is able to hold the NDCG performance. For all the three datasets, as the learning rate has reduced significantly, no successive clicks can change the NDCG performance, as shown in the graphs.

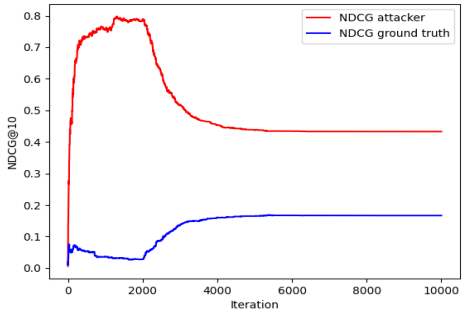
For further testing our hypothesis, we made the attacker attack in the later iterations. Figure 13 shows the results when the attack happens between 2000 and 4000 iterations. Figure 14 shows the results when the attack happens in 8000 to 10000 impressions. The results again follow the hypothesis; In Figure 13, we can see that the learning rate has already decreased from the initial setting. However, the learning rate has not decreased to a 0 value for MQ2007 and TD2003 datasets; attacking from here on-wards still improves the NDCG performance, although not as much as before. For MSLR and Yahoo dataset, the learning rate has already decreased to a very small value hence, the attack won't be able to improve any performance. A slightly different trend appears in figure 14, where the learning rate has already decreased to 0 for all the datasets. This means that even if the attacker tries to attack here, no further NDCG improvements will be there. Based on the above analysis, we can conclude that the attacker can attack the DBGD algorithm if it has a sensitive learning rate. Moreover, it is beneficial for the attacker to issue attacks as early as possible to ensure that the learning rate has not decreased significantly. On the other hand, if the learning rate is not sensitive, then the attacker has to keep attacking for a large number of iterations until the learning rate decreases to a small value, which is generally not feasible.



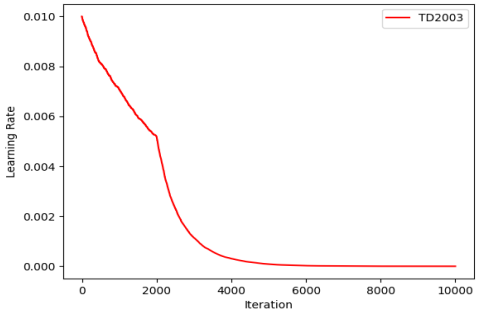
(a) Naive Intersection Attack MQ2007



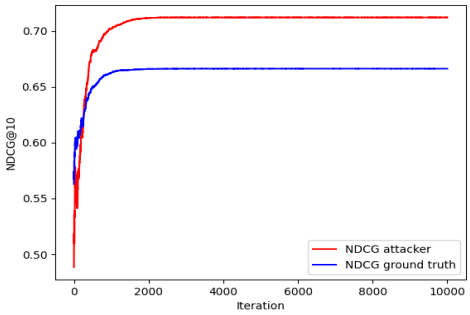
(b) Learning Rate MQ2007



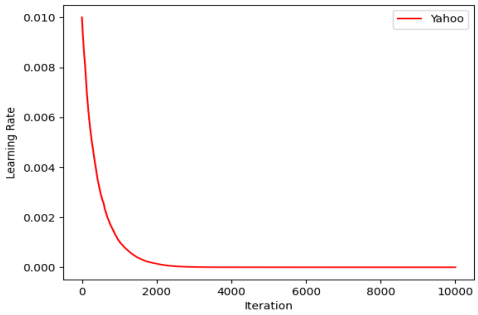
(c) Naive Intersection Attack TD2003



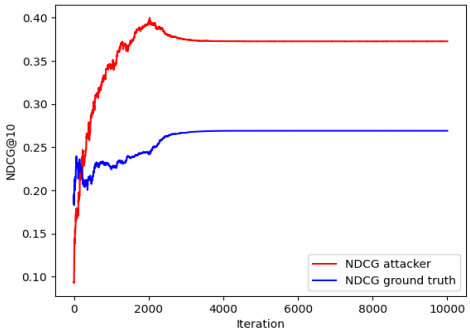
(d) Learning Rate TD2003



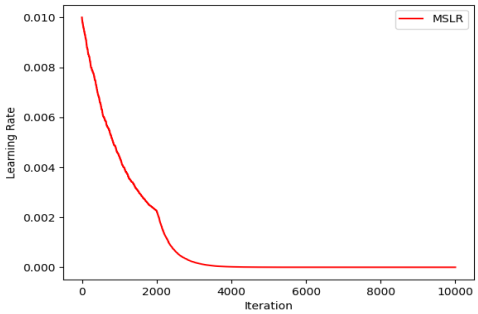
(e) Naive Intersection Attack Yahoo



(f) Learning Rate Yahoo

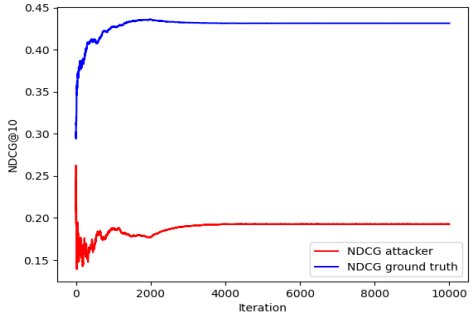


(g) Naive Intersection Attack MSLR

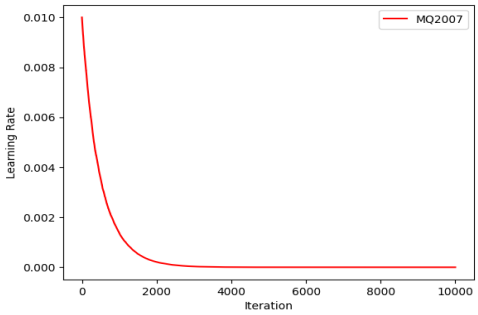


(h) Learning Rate MSLR

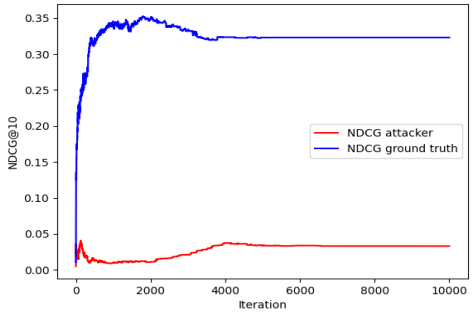
Figure 12: Reducing the learning rate decay



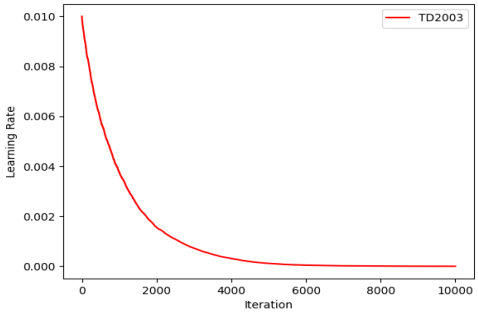
(a) Naive Intersection Attack MQ2007



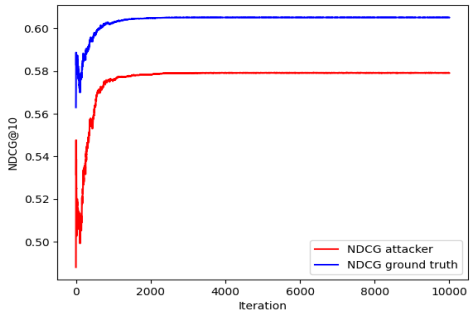
(b) Learning Rate MQ2007



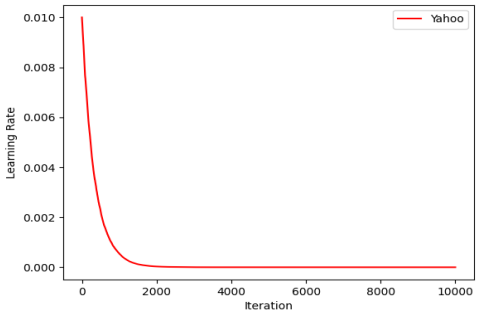
(c) Naive Intersection Attack TD2003



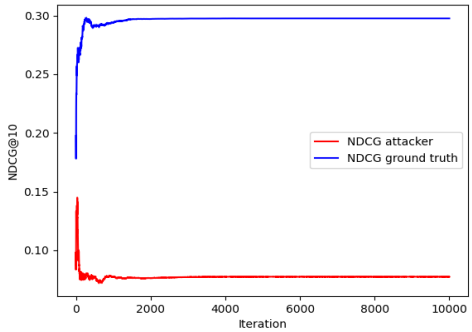
(d) Learning Rate TD2003



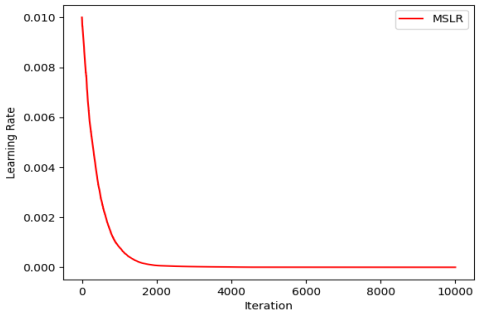
(e) Naive Intersection Attack Yahoo



(f) Learning Rate Yahoo

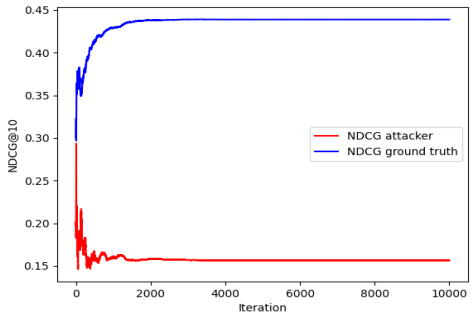


(g) Naive Intersection Attack MSLR

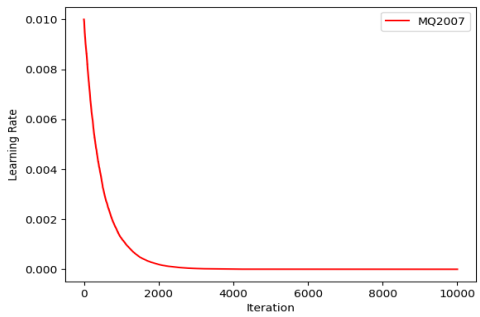


(h) Learning Rate MSLR

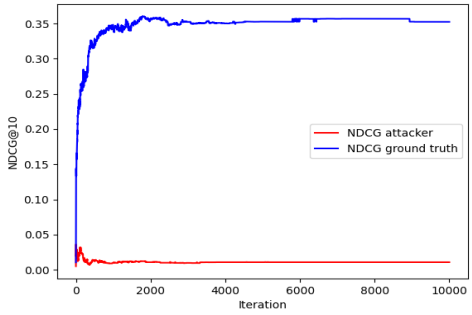
Figure 13: Attacking 2000 to 4000 impressions



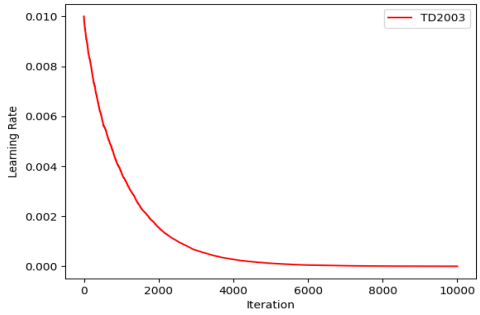
(a) Naive Intersection Attack MQ2007



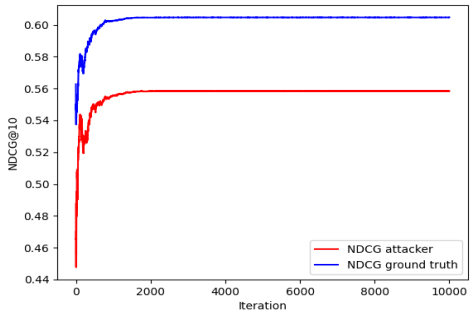
(b) Learning Rate MQ2007



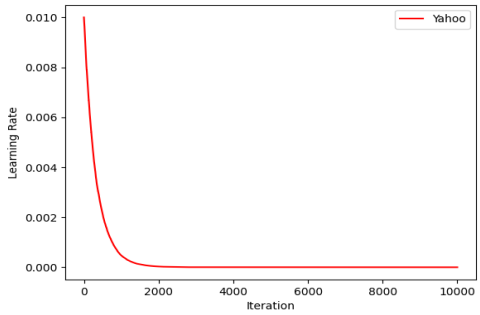
(c) Naive Intersection Attack TD2003



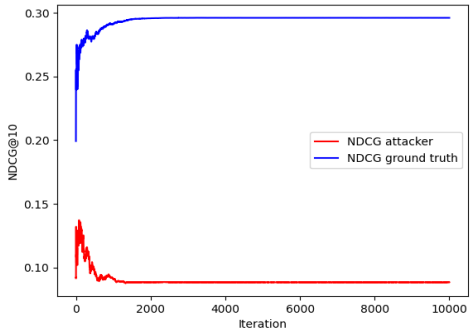
(d) Learning Rate TD2003



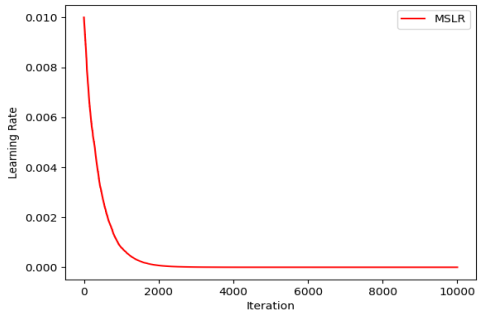
(e) Naive Intersection Attack Yahoo



(f) Learning Rate Yahoo



(g) Naive Intersection Attack MSLR

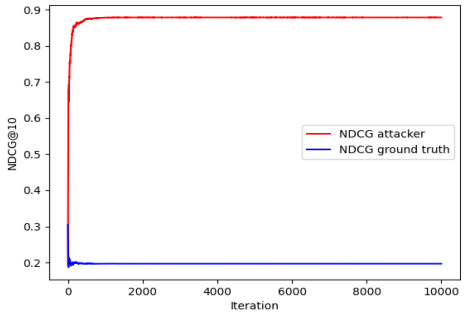


(h) Learning Rate MSLR

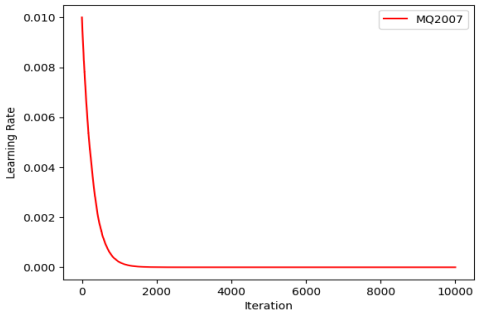
Figure 14: Attacking 8000 to 10000 impressions

6.2 Attacks on MGD

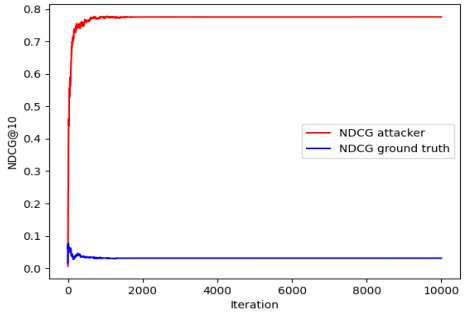
As explained previously, MGD is just a variation of DBGD where the algorithm can have multiple candidate rankers. Here we show the results on varying the number of candidates. Figure 15 shows the results when we have five candidate rankers, and Figure 16 shows the results with nine candidate rankers. In both cases, learning rate decay is 0.99, and only the first 2000 iterations are getting attacked. Since the MGD algorithm works faster than DBGD, we hypothesize that increasing the number of candidate rankers should quickly decrease the learning rate to 0. This happens because now there are multiple directions, and hence, there is a higher probability that whenever the attacker generates a click, the corresponding document comes from the exploratory ranker. Coherent with our hypothesis, it is shown that the learning rate decreases to a value of 0 at a much faster rate with 9 candidates, as compared to 5 candidates in all the datasets. Additionally, consistent with our previous hypothesis, the MGD can also hold the NDCG performance as the learning rate has decreased considerably after the attack. Following the above results we also show the effectiveness of frequency attack in the MGD based algorithms (Fig. 17). Again comparing with the frequency attack on DBGD based algorithms, a faster learning rate decrement is seen, hence showing that MGD based algorithms are more prone to the proposed attacks.



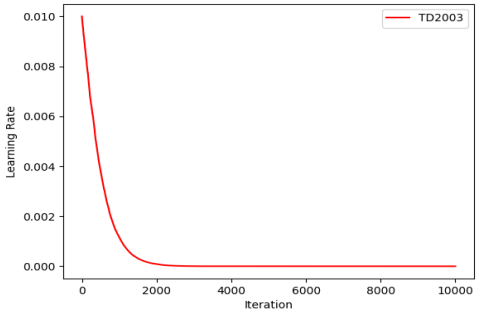
(a) Naive Intersection Attack MQ2007



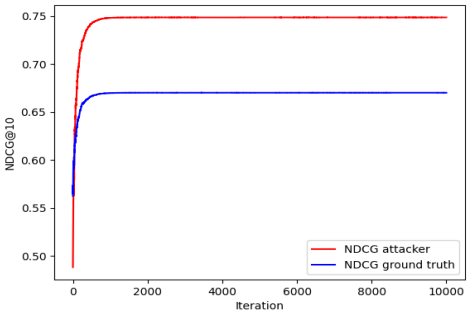
(b) Learning Rate MQ2007



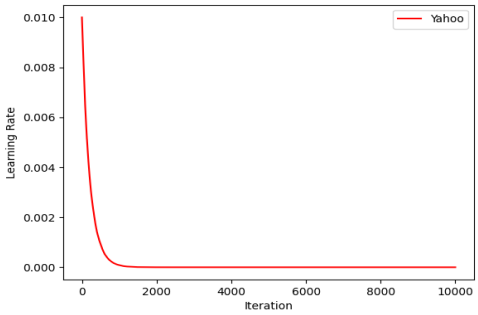
(c) Naive Intersection Attack TD2003



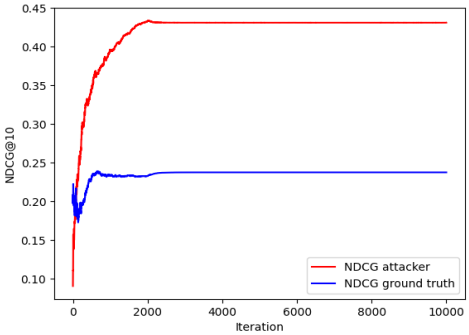
(d) Learning Rate TD2003



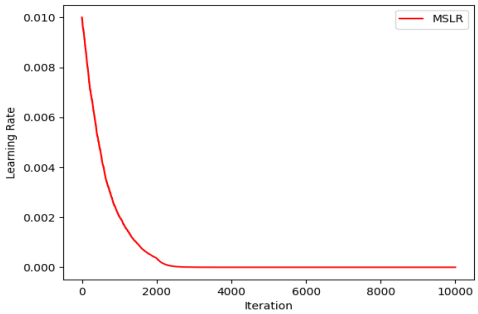
(e) Naive Intersection Attack Yahoo



(f) Learning Rate Yahoo

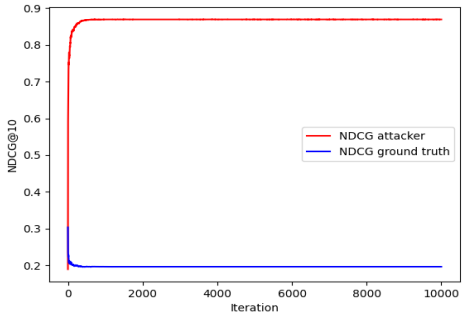


(g) Naive Intersection Attack MSLR

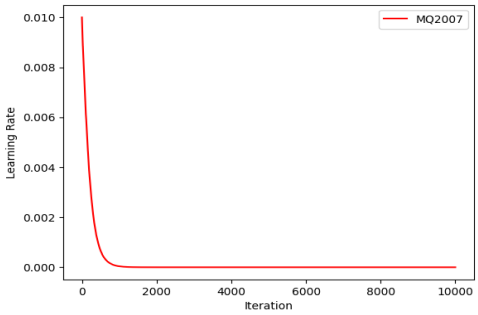


(h) Learning Rate MSLR

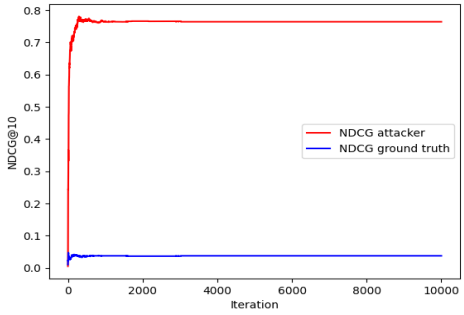
Figure 15: MGD with 5 candidates



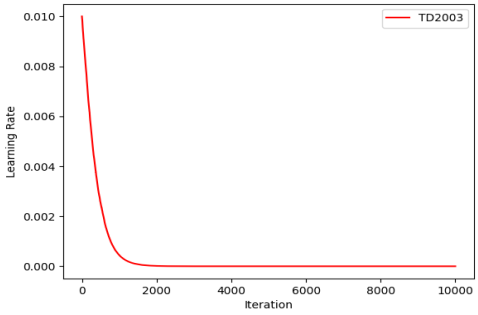
(a) Naive Intersection Attack MQ2007



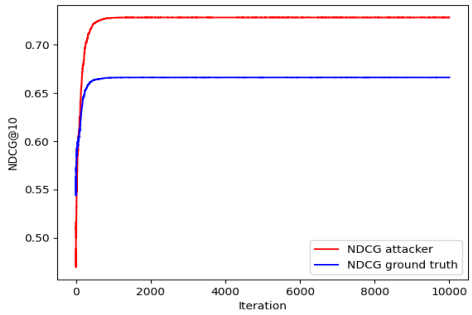
(b) Learning Rate MQ2007



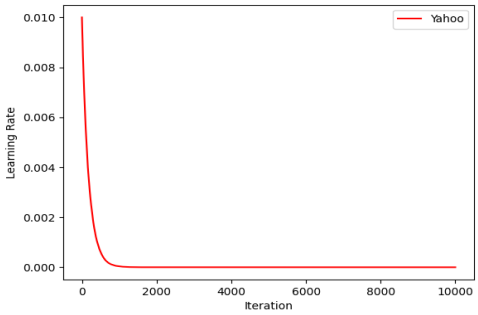
(c) Naive Intersection Attack TD2003



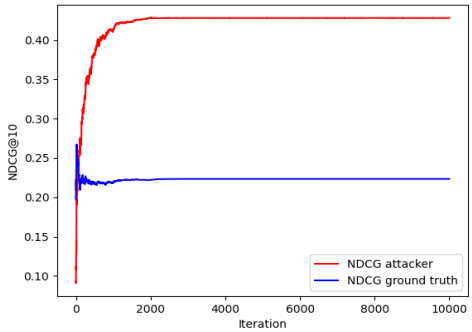
(d) Learning Rate TD2003



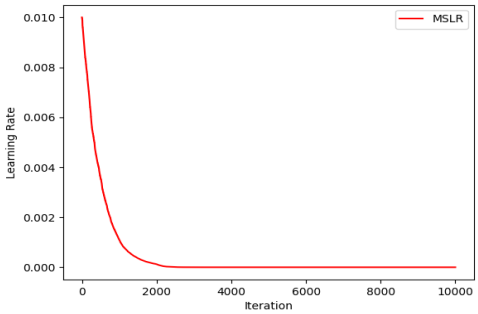
(e) Naive Intersection Attack Yahoo



(f) Learning Rate Yahoo

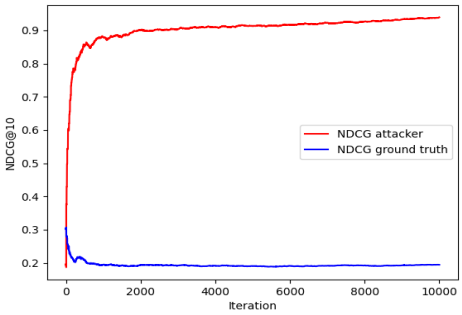


(g) Naive Intersection Attack MSLR

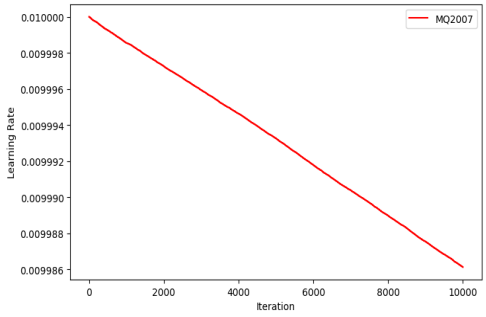


(h) Learning Rate MSLR

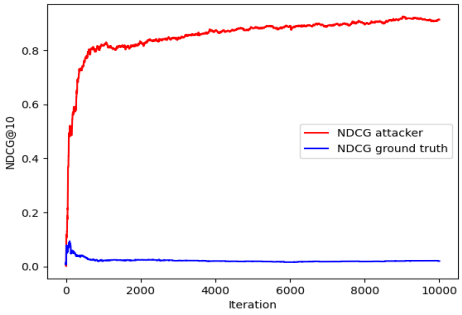
Figure 16: MGD with 9 candidates



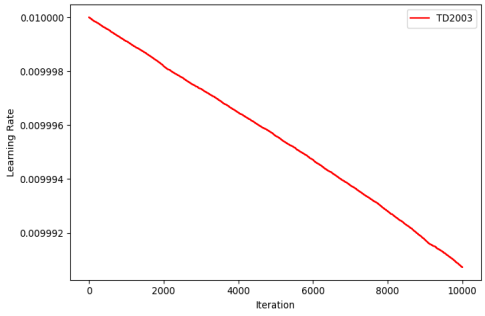
(a) Frequency Attack MQ2007



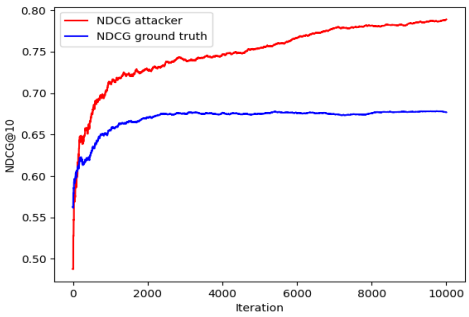
(b) Learning Rate MQ2007



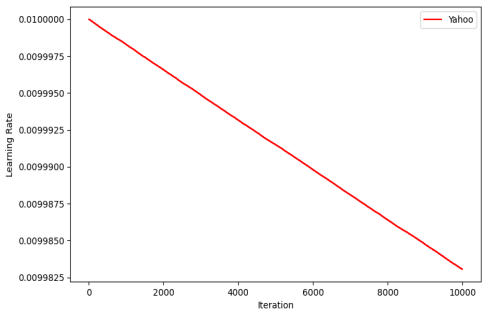
(c) Frequency Attack TD2003



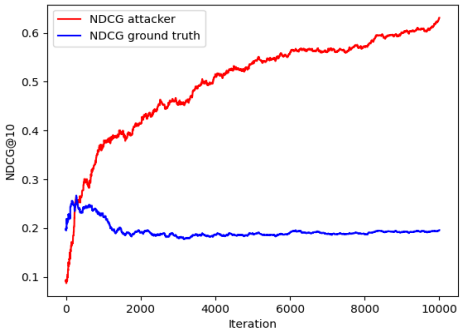
(d) Learning Rate TD2003



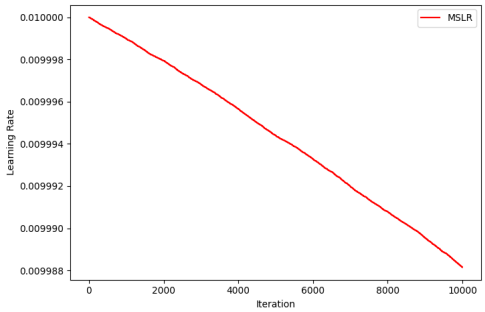
(e) Frequency Attack Yahoo



(f) Learning Rate Yahoo



(g) Frequency Attack MSLR



(h) Learning Rate MSLR

Figure 17: Effectiveness of Frequency attack on MGD

7 Conclusion and Future Work

In this thesis, we have explained the learning to rank problem, the traditional way to solve this problem, how it is being solved in modern times using OL2R, and why it has received more attention. The basics of DBGD and MGD are explained, and their problems are stated. This thesis aimed to present the flaws in DBGD and MGD based algorithms and show how an attacker can exploit these flaws to do poisoning attacks on the algorithms. Two different types of attacking strategies are presented, and the effectiveness of the attacks is shown for both DBGD and MGD. The results are further scrutinized, and in-depth analysis shows the effect of the learning rate behind it. We show that if the learning rate is sensitive, the model converges faster and becomes prone to attack. Moreover, increasing the number of exploratory rankers in MGD also affects the performance. We further conclude that if the attacker wants to succeed, it should attack the algorithms as early as possible with all the budget.

This research is still in very early phases. Despite showing the feasibility of the attacks, there are many limitations in our proposed framework. Firstly, we have assumed that the attacker has access to the training data, which might not be possible in many cases. In such scenario, the attacker cannot do targeted poisoning attacks but can still perform indiscriminate attacks. Secondly, the frequency attack also assumes that the attacker can query the IR system on the user's behalf, which is a rigid assumption and can only happen when the communication is severely exposed. Moreover, the hyper-parameters' fine-tuning was not done during this research as our main aim was to show the possibility of attacks and not the performance comparison between the attacks. Finally, the results we show are empirical results suggesting that the learning rate can heavily affect the attacks. A lot more in-depth theoretical research needs to be done in order to support this claim. Most of the emphasis of this thesis was to explain the robustness of DBGD and MGD based algorithms. We hope that this research might become a starting point for new researchers in the domain to re-think the possibility of attacks in such algorithms.

References

[1] Y. Yue and T. Joachims, "Interactively optimizing information retrieval systems as a dueling bandits problem," in *Proceedings of 26th ICML*. ACM, 2009, pp. 1201–1208.

[2] A. D. Flaxman, A. T. Kalai, and H. B. McMahan, "Online convex optimization in the bandit setting: gradient descent without a gradient," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2005, pp. 385–394.

[3] S. B. Kotsiantis, I. Zaharakis, and P. Pintelas, "Supervised machine learning: A review of classification techniques," *Emerging artificial intelligence applications in computer engineering*, vol. 160, no. 1, pp. 3–24, 2007.

[4] M. Sanderson *et al.*, "Test collection based evaluation of information retrieval systems," *Foundations and Trends® in Information Retrieval*, vol. 4, no. 4, pp. 247–375, 2010.

[5] X. Wang, M. Bendersky, D. Metzler, and M. Najork, "Learning to rank with selection bias in personal search," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016, pp. 115–124.

[6] P. Vakkari and N. Hakala, "Changes in relevance criteria and problem stages in task performance," *Journal of Documentation*, vol. 56, no. 5, pp. 540–562, Oct. 2000. [Online]. Available: https://doi.org/10.1108/eum0000000007127

[7] A. Schuth, H. Oosterhuis, S. Whiteson, and M. de Rijke, "Multileave gradient descent for fast online learning to rank," in *Proceedings of the Ninth ACM International Conference on WSDM*. ACM, 2016, pp. 457–466.

[8] K. Hofmann, S. Whiteson, and M. de Rijke, "Balancing exploration and exploitation in listwise and pairwise online learning to rank for information retrieval," *Information Retrieval*, vol. 16, no. 1, pp. 63–90, Apr. 2012. [Online]. Available: https://doi.org/10.1007/s10791-012-9197-9

[9] T. Joachims, "Optimizing search engines using clickthrough data," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, 2002, pp. 133–142.

[10] F. Radlinski, M. Kurup, and T. Joachims, "How does clickthrough data reflect retrieval quality?" in *Proceedings of the 17th ACM CIKM*. ACM, 2008, pp. 43–52.

[11] B. L. Pereira, A. Ueda, G. Penha, R. L. Santos, and N. Ziviani, "Online learning to rank for sequential music recommendation," in *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019, pp. 237–245.

[12] A. Newell, R. Potharaju, L. Xiang, and C. Nita-Rotaru, "On the practicality of integrity attacks on document-level sentiment analysis," *Proceedings of the ACM Conference on Computer and Communications Security*, vol. 2014, pp. 83–93, 11 2014.

[13] K. Auernhammer, R. T. Kolagari, and M. Zoppelt, "Attacks on machine learning: Lurking danger for accountability," in *SafeAI@AAAI*, 2019.

[14] M. Comiter, B. C. for Science, and I. Affairs, *Attacking Artificial Intelligence: AI's Security Vulnerability and what Policymakers Can Do about it*, ser. Belfer Center paper. Belfer Center for Science and International Affairs, 2019. [Online]. Available: https://books.google.com/books?id=bUUkygEACAAJ

[15] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *arXiv preprint arXiv:1702.02284*, 2017.

[16] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv 1412.6572*, 12 2014.

[17] Y.-C. Lin, Z.-W. Hong, Y.-H. Liao, M.-L. Shih, M.-Y. Liu, and M. Sun, "Tactics of adversarial attack on deep reinforcement learning agents," *arXiv preprint arXiv:1703.06748*, 2017.

[18] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[19] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli, "Is feature selection secure against training data poisoning?" in *International Conference on Machine Learning*. PMLR, 2015, pp. 1689–1698.

[20] S. Mei and X. Zhu, "Using machine teaching to identify optimal training-set attacks on machine learners," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, 2015.

[21] J. Wang, Y. Song, T. Leung, C. Rosenberg, J. Wang, J. Philbin, B. Chen, and Y. Wu, "Learning fine-grained image similarity with deep ranking," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1386–1393.

[22] Z. Niu, M. Zhou, L. Wang, X. Gao, and G. Hua, "Hierarchical multimodal lstm for dense visual-semantic embedding," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 1881–1889.

[23] J. Li, R. Ji, H. Liu, X. Hong, Y. Gao, and Q. Tian, "Universal perturbation attack against image retrieval," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 4899–4908.

[24] Z. Liu, Z. Zhao, and M. Larson, "Who's afraid of adversarial queries? the impact of image modifications on content-based image retrieval," in *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, 2019, pp. 306–314.

[25] X. Liu, S. Si, X. Zhu, Y. Li, and C.-J. Hsieh, "A unified framework for data poisoning attack to graph-based semi-supervised learning," *arXiv preprint arXiv:1910.14147*, 2019.

[26] X. Zhang, X. Zhu, and L. Lessard, "Online data poisoning attacks," in *Learning for Dynamics and Control*. PMLR, 2020, pp. 201–210.

[27] M. Melucci, *Vector-Space Model*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009.

[28] A. Schuth, F. Sietsma, S. Whiteson, D. Lefortier, and M. de Rijke, "Multileaved comparisons for fast online evaluation," in *Proceedings of the 23rd ACM CIKM*. ACM, 2014, pp. 71–80.

[29] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *Arxiv*, 06 2012.

[30] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[31] K. You, M. Long, J. Wang, and M. I. Jordan, "How does learning rate decay help modern neural networks?" 2020. [Online]. Available: https://openreview.net/forum?id=r1eOnh4YPB

[32] G. Yang, N. Z. Gong, and Y. Cai, "Fake co-visitation injection attacks to recommender systems." in *NDSS*, 2017.

[33] A. R. Chordiya, S. Majumder, and A. Y. Javaid, "Man-in-the-middle (mitm) attack based hijacking of http traffic using open source tools," in *2018 IEEE International Conference on Electro/Information Technology (EIT)*. IEEE, 2018, pp. 0438–0443.

[34] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. 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*, vol. 310, 2007.

[35] T. Qin and T. Liu, "Introducing LETOR 4.0 datasets," *CoRR*, vol. abs/1306.2597, 2013. [Online]. Available: http://arxiv.org/abs/1306.2597

[36] O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview," in *Proceedings of the Learning to Rank Challenge*, 2011, pp. 1–24.

[37] H. Oosterhuis and M. de Rijke, "Differentiable unbiased online learning to rank," in *Proceedings of the 2018 ACM on Conference on Information and Knowledge Management*. ACM, 2018.

[38] F. Guo, C. Liu, and Y. M. Wang, "Efficient multiple-click models in web search," in *Proceedings of the Second ACM International Conference on WSDM*. ACM, 2009, pp. 124–131.