Graduate School Form
30 Updated 12/26/2015

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By HAREESH SANDUPATLA

Entitled

USING REINFORCEMENT LEARNING TO LEARN RELEVANCE RANKING OF SEARCH QUERIES

For the degree of Master of Science

Is approved by the final examining committee:

Dr. Mohammad Al Hasan
Chair

Dr. Rajeev Raje

Dr. Snehasis Mukopadhyay

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy of Integrity in Research" and the use of copyright material.

Approved by Major Professor(s): Dr. Mohammad Al Hasan

Approved by: Dr. Shiaofen Fang 4/13/2016
Head of the Departmental Graduate Program Date

USING REINFORCEMENT LEARNING TO LEARN RELEVANCE RANKING OF SEARCH QUERIES

A Thesis

Submitted to the Faculty

of

Purdue University

by

Hareesh Sandupatla

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

May 2016

Purdue University

Indianapolis, Indiana

To my parents for all their support, love and providing me in the best education possible. I am grateful to my whole family for their sacrifices. Without them, I wouldn't have been to this far.

ACKNOWLEDGMENTS

First and foremost, I am grateful to God. Without his blessings, I feel I couldn't gain the knowledge and stay patient to complete this thesis. My faith in God gave me the strength to believe in myself and wisdom to deal with difficulties. My prayers to God have always helped me throughout my life.

I would also like to take this opportunity to thank my supervisor Dr. Mohammad Al Hasan for not only having the trust on me but also for guiding and encouraging me throughout the research. His suggestions and insights have helped me to gain the comprehensive knowledge of the research topic. I greatly admire him for his knowledge, patience, dedication and passion towards the work. In spite of a great professor, he is also a good human being who is very kind at heart and shows empathy to the student's feelings.

Besides my advisor, I would like to express my profound gratitude to Dr. Rajeev Raje and Dr. Snehasis Mukhopadhyay for graciously agreeing to be a part of my thesis advisory committee. Their guidance helped me to take wise decisions during my research and studies. For many years, they have been sharing a lot of knowledge and experiences with the IUPUI students. So, I feel so privileged to present my work in front of them.

I am also thankful to my professors Dr. Xukai Zou, Dr. Yao Liang and Arjan Durresi whose courses have helped me to improve the knowledge of Computer Science. I appreciate the pleasant smiles and warmth greetings that I had exchanged with other professors at the Computer Science department and during various seminars. A special thanks to the Computer Science staff Nicole Wittlief, Scott Orr and Nancy Reddington. The support, knowledge and guidance you all have provided for your students is priceless.

A special thanks to my dear friends and fellow lab-mates Tanay Kumar Saha, Mahmudur rahman, Vachik Dave, Baichuan Zhang and Manusurul Alam whose invaluable help, on many levels, has made my work possible. I won't forget the kind of support I got from them during the research especially during my struggle to gain familiarity with Latex and Linux Operating System. I am also grateful to them for the discussions and the pleasure we have had in the lab.

I always had a big thanks in my heart for my parents, brother and sister. I thank them for their unlimited love and support. They always had faith in me and encouraged me to reach my highest possibilities particularly when things got overwhelmed. They are my source of love and support even when they are away.

TABLE OF CONTENTS

Page

LIST OF FIGURES

ABSTRACT

Sandupatla, Hareesh. M.S., Purdue University, May 2016. Using Reinforcement Learning to Learn Relevance Ranking of Search Queries. Major Professor: Mohammad Al Hasan.

Web search has become a part of everyday life for hundreds of millions of users around the world. However, the effectiveness of a user's search depends vitally on the quality of search result ranking. Even though enormous efforts have been made to improve the ranking quality, there is still significant misalignment between search engine ranking and an end user's preference order. This is evident from the fact that, for many search results on major search and e-commerce platforms, many users ignore the top ranked results and click on the lower ranked results. Nevertheless, finding a ranking that suits all the users is a difficult problem to solve as every user's need is different. So, an ideal ranking is the one which is preferred by the majority of the users. This emphasizes the need for an automated approach which improves the search engine ranking dynamically by incorporating user clicks in the ranking algorithm. In existing search result ranking methodologies, this direction has not been explored profoundly.

A key challenge in using user clicks in search result ranking is that the relevance feedback that is learnt from click data is imperfect. This is due to the fact that a user is more likely to click a top ranked result than a lower ranked result, irrespective of the actual relevance of those results. This phenomenon is known as position bias which poses a major difficulty in obtaining an automated method for dynamic update of search rank orders.

In my thesis, I propose a set of methodologies which incorporate user clicks for dynamic update of search rank orders. The updates are based on adaptive random-

ization of results using reinforcement learning strategy by considering the user click activities as reinforcement signal. Beginning at any rank order of the search results, the proposed methodologies guaranty to converge to a ranking which is close to the ideal rank order. Besides, the usage of reinforcement learning strategy enables the proposed methods to overcome the position bias phenomenon. To measure the effectiveness of the proposed method, I perform experiments considering a simplified user behavior model which I call color ball abstraction model. I evaluate the quality of the proposed methodologies using standard information retrieval metrics like Precision at n ($P@n$), Kendall tau rank correlation, Discounted Cumulative Gain (DCG) and Normalized Discounted Cumulative Gain ($NDCG$). The experiment results clearly demonstrate the success of the proposed methodologies.

1 INTRODUCTION

With tremendous growth of information in web, measuring and improving the quality of the retrieval system has become an absolute necessary. The initial studies of measuring retrieval systems performances has generally based on the Cranfield methodology [1], which relies on explicit relevance judgement collected from human experts. But unfortunately gaining explicit relevance judgement is expensive both in terms of time and effort. Because of this limitation, the methods based on explicit relevant judgements are difficult to apply to the commercial search engines that have large scale search services. Moreover, in some domains like medical search [2], it is not feasible to gather relevance judgement. Furthermore, it has been shown that the metrics based on human judgements have no significant correlation with user-centric performance measures [3].

Lack of effectiveness and excessive cost of explicit feedback have encouraged the information retrieval scientists to use implicit relevance feedback for measuring the quality of a retrieval system. Such feedback comes in the form of user's day to day interactions with the search interface. For all commercial search engines, these interactions are generally captured in the web logs. The most important form of user interaction is user clicks, also known as *click-through* data. Click-through data provides non-trivial implicit feedback of the quality of search result ranking [4]. Thus, using this feedback as relevance judgement provides access to virtually unlimited amounts of data for assessing and optimizing the performance of information retrieval systems.

A trivial approach of using click-through data to improve search result ranking is to arrange the results in decreasing order of the number of clicks that a result receives. Here, the assumption is that given an arbitrary pair of search results, the one that has higher relevance receives more clicks. However, this assumption is not

necessarily true for most of the real world search engines, because user clicks on a result also depend on the position of that result in the search result ranking. So, any method that leverages click-through data as implicit feedback for improving search result ranking must consider user's click behavior into account.

Most of the users don't click the search results randomly. Rather they deliberately choose the results to click after careful examination. To understand user's click behavior in a search session, researchers have performed eye tracking experiments [5]. These experiments suggest that a user scans the search results from top to bottom and clicks a result which he feels the most relevant among all the results that he has viewed. For example, if the user clicks on the second result after ignoring the first result, we can deduce that the relevance of the second result is greater than that of the first result. In this way, user clicks do not provide an absolute relevance judgement, but they accurately provide the relative judgement between different search results. Earlier works also proved that, for a given query, for a specific result, the probability of click on that result decays with it's position in the rank order. In other words, the probability of click is not only influenced by the relevance of a result but also on its rank in the search result page. The user's behavior of clicking a higher ranked result more often than a lower ranked result is known as *position bias* .

Previously, some researchers have made efforts to build the ranking system using click-through data [5]. They have utilized supervised or semi-supervised machine learning techniques to implicitly construct a ranking model using a given training data. This approach is also referred as *learning to rank* technique. This learning to rank method has been effectively applied to Web search and Information Retrieval domain.

For example, Joachims [5] has proposed a ranking method that use Support Vector Machine (SVM). In the proposed learning to rank approach, he formulated the ranking process as the problem of learning with relative preferences are given as input. For example, "for the given pair of documents (d_i, d_j), d_i should be ranked higher to d_j with respect to a specific query q". These pairwise preferences can be inferred

from the user click patterns. This is a classic pairwise learning to rank method. However, this is an off-line approach. This approach needs the information of user's past activities (click-through data) to train the ranking algorithm.

Kemp and Ramamohanrao [6] have also proposed a method of learning to rank using the click-through data. In their method, they represented every document and query as a vector using the vector space model. Their learning method is based on *Document Transformation*. The document transformation is an idea of moving document vector towards a query which is known to be relevant to that document. Their experiments have suggested that document transformation improves retrieval performance over large collections of documents. However, this is an offline method too.

Although enormous efforts have been made to improve the ranking quality using the user click-through data, there is still significant misalignment between search engine ranking and an end user's preference order. This is evident from the fact that, for many search results on major search and e-commerce platforms, many users ignore the top ranked results and click on the lower ranked results. Nevertheless, finding a ranking that suits all the users is a difficult problem to solve as every user's need is different. So, an ideal ranking is the one which is preferred by the majority of the users. This emphasizes the need for an automated approach which improves the search engine ranking dynamically by incorporating user clicks in the ranking algorithm.

In my thesis work, I address this issue of misalignment between search engine ranking and end user ranking. My thesis work helps to improve the search engine ranking by resolving these misalignments. I propose a set of methodologies which incorporate user clicks for dynamic update of search rank orders. The updates are based on adaptive randomization of results using reinforcement learning strategy by considering the user click activities as the reinforcement signal. Beginning at any rank order of the search results, the proposed methodologies guaranty to converge to a ranking which is close to the ideal rank order. Besides, the usage of reinforce-

ment learning strategy enables the proposed methods to overcome the position bias phenomenon. To measure the effectiveness of the proposed methods, I perform experiments considering a simplified user behavior model which I call color ball abstraction model.

2 RELATED WORKS

The advent of computers and development of world wide web have made it is possible to store huge volumes of information in the internet. At the same time people are becoming more and more dependent on Internet for their information need. But, considering the gigantic use of Internet, it is certainly unrealistic to expect an ordinary user to identify the required information by simply browsing the web. As a result, it has become a compulsion to identify an easy and efficient method to retrieve the information from the web. This lead us to a new research area called Information Retrieval (IR).

The Ranking is a key problem for Information Retrieval (IR). The process of ranking the results is implicitly embedded in many IR problems such as key term extraction, definition finding, sentiment analysis, collaborative filtering, document retrieval, web spam detection, product rating and important email routing. In my thesis, I primarily focus on document retrieval. In the document retrieval, a set of documents is given for ranking. The ranking system takes a query as the input and evaluates the documents to generate a score for every document, which is a criterion for ranking the documents. In IR literature, many heuristic ranking methods are proposed to handle the problem of document retrieval. In order to summarize these models, I perform a categorization of these IR models.

2.1 Traditional Content Based Models

The early IR ranking models are based on the similarity between the query terms to the document. Examples include *Vector Space Model* (VSM) [7] and *Latent Semantic Indexing* (LSI) [8]. In the VSM model, queries and documents are represented as vectors in the Euclidean space. The vector representation of documents and queries

can be achieved using TF-IDF (Term Frequency and Inverse Document Frequency). The VSM model has a limitation that it assumes independence between the query terms which is a major drawback of this approach. This limitation has made the model unrealistic and inefficient. A user has various ways to express a given concept in text. The users want to retrieve the information on the basis of overall conceptual content of query rather than the meaning of independent word. The isolated words may provide uncertain manifestation of the concept or meaning of a document or a query.

The LSI model has been developed to over come the flaw of the term-matching based information retrieval by treating the unreliability of observed term-document relationship data as a statistical problem. They took the advantage of implicit latent structure in the association of terms with documents using the *Singular Value Decomposition* (SVD). They took a large matrix of term-document association data and constructed a "semantic" space wherein terms and documents that are greatly related are placed close to one another. The SVD permits the arrangement of the space to demonstrates the vital associative patterns in the data, and ignores the trivial influences. As a result, the terms that did not actually appear in a document may still become highly related to the document, if that is consistent across the major patterns of association in the data. Position in the semantic space then serves as the new source for indexing, and retrieval process continues by using the terms in a query to identify a point in the semantic space, then the documents in its corresponding neighborhood space are returned to the user.

Thereafter, the researches have shown greater attention towards the models based on the probabilistic ranking principle [9] such as *BM25* and *language model for IR*. These models uses the technique of *Probabilistic Indexing* . For a give request for information, the Probabilistic Indexing process provides a statistical inference which allows to find a relevance of each document to a given query. This relevance value is a measure of the probability that a document fulfils the given query. The key feature of BM25 [10] ranking model is to rank the given documents according to log-odds of

their relevance score. The language model for IR is a relatively common conventional approach with different versions. The common basic approach for using language model in IR is based on query likelihood model. In this approach the researchers constructed a language model from each document in the given collection. Then they ranked the documents according to query likelihood value. The query likelihood value is a probability generated using the terms of the query based on language model. Thereafter, many variants of language models for IR have been proposed. Some of them are based on content similarity [11], topic diversity [12], K-L divergence [13] and hyperlink structure [14].

2.2 Relevance Feedback Models

I broadly categorize the studies of measuring the retrieval system performance using relevance feedback methods into following two types, namely, Explicit relevance feedback Models and Implicit relevance feedback Models

2.2.1 Explicit Relevance Feedback Models

These models are the pioneers of relevance feedback methods. The initial studies of measuring retrieval systems performances have commonly used Cranfield methodology (many tasks are discussed in [1]), which is based on relevance judgements provided manually by human experts. In this approach, each query has a label that defines the relevance of each document as per a graded relevance scale. Given a ranking produced by a retrieval system and the corresponding query, the retrieval system quality can be assessed by aggregating the judgement of the top ranked documents. Metrics such as $NDCG$ (Normalized Discounted Cumulative Gain), Mean Average Precision and Precision@n [15] can be used to aggregate over many queries yielding the overall performance score.

Thereafter the relevance feedback raking has been relayed on hand designed ranking functions like [16]. But later hand designing of ranking functions have become

extremely difficult and intractable with the addition of thousands of features for ranking. Many machine learning algorithms have been applied for optimization of the ranking function such as [17] [18]. But, huge number of training examples with relevance labels are required to train the ranking functions based on machine learning algorithms. The cost of generating these examples is very expensive. Besides, for the time sensitive queries such as "President of United States", relevance of document is not fixed and varies over the time.

In 1994 Bartell et al. [19] proposed an approach based on mixture-of-experts by which the relevance estimates made by different experts could be combined to provide the superior retrieval performance. Their method of combining different rankings is based on the gradient-descent technique. In this approach the rankings have seen as real valued scores and the problem of combining different rankings has modeled as a numerical search for a collection of parameters that minimize the discrepancy between the combined scores and the feedback of expert. However, this approach is also based on explicit relevance judgment.

A similar method was proposed by Cohen et al. [20]. In their approach, they constructed the preference graphs based on rankings and the problem of ranking has modeled as a combinatorial optimization problem. The formulation is NP-complete. Hence, they used an approximation to combine different rankings. They have demonstrated experimentally and theoretically that their method finds a combination that performs near to the best of basic experts.

However, explicit feedback is expensive to collect. In real world setting, most of the users are unwilling to give such feedback as this need additional effort. This makes the labeled dataset small, and inadequate to work. Moreover in some domains, such as, medical search [2], it is not feasible to gather relevance judgements. Furthermore, some metrics based on human judgements such as Mean Average Precision (MAP) and Precision at 10 documents (P@10) have been shown to not essentially correlate with user-centric performance measures [3]. In 2006, Turpin et al. have investigated user performance based on two simple tasks namely precision-oriented task, recall-

based task. Their experimental results proves that there is no significant relationship between system effectiveness evaluated by precision based task with user performance evaluated by the recall based task.

2.2.2 Implicit Relevance Feedback Models

Eventually, to overcome the drawbacks of explicit feedback, many researchers have concentrated on ranking using the data collected implicitly from the user such as user click-through data, time spent on particular page and activities like printing and bookmarking/annotation. A click-through is a link click event, when he see the retrieved documents of their search endeavor. This approach is based on the hypothesis that the documents are more likely to be relevant if the user click on it. Click-through data embed important information about the user satisfaction. It is possible to enhance the performance of information retrieval systems by taking and investigating the user past activities. Every day number of people interact with web search engine which eventually generates enormous amount of user click-through data containing vital implicit feedback.

Even though we have ample click-through data we must take care while leverage it as implicit feedback, since user behavior is impacted by document's order. Clicks-through data is biased by the presentation order of the documents. Understanding how user access the search results is the key aspect to deduce implicit preferences from click-through log-files. Because we can only extract valid implicit feedback for results that user has literally observed at and assessed. So, better understanding of a user behavior will allow us to get more factual inferences about relating implicit feedback with relevance judgements.

The primitive research works related to learning the user behavior was based on passively recorded user navigation actions namely hyper-links clicked and hyper-links passed over. In [21] [22] they have used these behavior as the factors to evaluate the user interest in a page. But in [23], Jeremy Goecks and Jude Shavlik have attempted

to learn the user's interest by unobtrusively observing his actions like user mouse scrolling actions along-with the browsing activity. They have built an agent based on the standard neural network to find the user's interests in the World Wide Web. Their experiment suggests that the agent can learn to predict accurately.

In 1994, Mortia and Shinoda [24] have estimated the interrelationship between reading time and user interest. And they inferred that user interest levels can be found using the amount of time they have spent on reading the page. They deduct this by experimenting on USenet news articles readers. Advocating this, in 1997 [25] Konstan has showed that user's interest levels could be greatly influenced by reading time. In their GroupLens system, they showed that reading time is the strong factor to deduct the user's interest level.

In 1998, Douglas W. Oard and Jinmook Kim [26] have identified different sources of implicit feedback and provided the methods to use them in recommender systems. First category is "examination" which seeks to get the ephemeral interactions throughout a session. Second category is "retention", this group of user behaviors which recommend an intention for use of material in future. Third category is "reference", which has the group of user behavior that construct explicit and implicit links between information objects.

In 2004, Laura et al. [27] investigated how users interact with the result page of a WWW search engine using eye-tracking. The key insight of their work is to understand how the users browse the exhibited abstracts and how they click links for further inspection. They performed an experiment with 36 participants. All participants were undergraduate students of different majors at a large university in Northeast USA. They have given 10 questions and answers. 5 among them are homepage searches, and rest of them are informational searches [28]. The questions have chosen from various difficulty levels and topics, namely science, movies, local politics, college etc .

The first aspect of their eye-tracking experiment is to find how does rank influences the fixation time. From their experiment they have inferred the following facts. For

the abstracts presented at rank 1 and rank 2, the mean time the users fixate on the abstract is approximately the same. But this contradict the known fact that users considerably more often click on the link at rank 1. The fixation rate drops down abruptly for the abstracts after rank 2. For the abstracts at rank 6 and rank 7, the fixation time and the number of clicks are approximately the same. For the abstracts at rank 6 to 10, each abstract has received almost equal attention which is not the case with rank 1 to 5. This is because of the fact that usually only the first 5-6 abstracts were appeared without scrolling down. So, once the user has scrolled down, the document order is insignificant for user attention.

The second valuable aspect is to understand how does an user explore the list. While leverage the user actions as implicit feedback about performance of a retrieval system, it is essential to understand how diligently users have evaluated the exhibited results before making any selection. To understand how the users explore the list, they have determined how many results above and below the clicked result users scan on average. They have identified that the number of links observed below a click is low after rank 1, which means that users have scanned the list from top to bottom. comprehensively, the users who clicked lower ranked results viewed proportionately more abstracts.

In [4], the authors have attempted to build a model of query-dependent deviations using implicit feedback. Their model of click-through interpretation has better prediction accuracy over contemporary standard click-through models. In their work, they have derived 4 different models for predicting user preferences namely Baseline Model, Click-through Model and General User Behavior Model. The performance of their model wasn't the same for all queries. For example, performance was poor for queries with multiple meanings. So this approach has given new direction of clustering the queries and learning different predictive models for each group of queries.

Kemp et al. [6] presented an approach based on document expansion and transformation for learning from a search engine. They have used click-through data for document expansion. The documents clicked are highly relevant to the query, they

assumed. They expanded the document by adding the query words to the document clicked. Their work advocates that altering the content of indexed document in relation to the past selection behavior as a means to keep the documents much closer to corresponding queries. Indexed term vector of document are added with the query terms for which a document got selected. This action is similar to imperatively weighing those terms in document. Subsequently, this directs the document to drift towards the query terms for which it was selected before. They have shown that document transformation can be used to reinforce retrieval performance over large set of web pages.

In [5], Thorsten Joachims has presented an approach which takes the click-through logs to train the search engine rankings, based on Support Vector Machines. Here, the key aspect is that the click-through data can provide the information of relative preferences. They have formulated the problem of learning ranking function over a finite domain with regards to the risk minimization. They have presented a raking SVM algorithm which leads to convex program which can provide non-linear ranking functions. Their experiments shows that raking SVM approach can successfully learn the most efficient learning function.

Later, many researchers have made an attempt to model user click behavior during search. Their idea is to accurately predict the future clicks employing the past click-through logs. Broadly, these click models can be categorized into two types, such as the "position model" [29] [30] and the "cascade model" [29] [31]. The assumption in position model is that a click depends on its relevance and examination as well. So if a document is clicked means the document is examined and felt relevant by the user. The cascade model assumes that user examines the results from top to bottom and finishes the search once relevant document is clicked. In this model the probability of examination depends on both rank and relevance of previous documents. Both of them have their own pro and cons. The position model deals the different documents in a search results page independently. So this model fails to seize the interaction among the documents in the examination probability. The cascade model has a strong

assumption that there is only one click per search so it fails to explain search having more than one clicks.

None of the above user click behavior models has distinguished the perceived relevance and actual relevance. In 2009, Olivier Chapelle [32] has proposed a Dynamic Bayesian Network based approach which provides the unbiased estimates from the click-through logs. Their model is different from the previous ones in two aspects. They assumed that click doesn't essentially signifies that the user is satisfied with it, so they have accounted both perceived relevance and actual relevance. They also didn't limit the number of clicks in a search.

In 2006 Radlinski and Joachims [33] has proposed an approach based on modification to search result presentation which gives relevance judgements that are not impacted by the presentation bias under some rational assumptions. They have build a model which is based on two assumptions. First assumption states that user continues his search endeavor until he finds any sufficiently relevant document. That also means that user doesn't skip over a result if he recognizes and finds it to be relevant. Let me define a few terms namely Item Relevance Score and Ignored Relevance Score before describing the second assumption. In a particular ranking, if a particular document $d1$ is replaced with a less relevant document $d2$ while keeping remaining all unchanged, the difference between the probability of $d1$ being selected and the probability that $d2$ being selected is the Item Relevance Score. In a particular ranking, if a particular document $d1$ is replaced with the more relevant one $d2$ while keeping remaining all unchanged, the difference between the probability of the user selecting the next document (after the replaced one) and the probability without the change is the Ignored Relevance Score. The second assumption they made is that Ignored Relevance Score is smaller than Item Relevance Score.

They have presented an algorithm based on FairParis. The main idea of their algorithm is to randomize part of the presentation order to remove the impact of presentation bias with the help of minimum number of modifications to the ranking. Their idea is to divide the ranking documents into a set of FairPairs. Then each

pair of results are considered independently, and flipped with 50 percent probability. Then they take this result set and present to the user and record the clicks on each document. The key idea in this process is that half of the documents will be presented in the original rank, and all documents will be presented together one rank from the original rank. Every time the lower document in a pair that has considered for flipping is clicked means the lower ranked document will be preferred over the one above one. They also proved that learning with data from FairPairs will converge to an ideal ranking if the ranking exists.

3 COLOR BALL ABSTRACTION MODEL

In World Wide Web search scenarios, user behavior is complicated. Creating such user behavior model in the lab to analyze search ranking is nearly impossible. This led us to built a simplified user behavior model which I refer as *color ball abstraction model*. The details of this user behavior model are given in subsequent paragraphs.

In a typical web search endeavor, for a specific query, every user's need is different. The result expected by one user may not necessarily match with the expectation of other user. So, the best ranking is to order the results in such a way that number of users who expect the top ranked result is greater than that of a lower ranked result. This suggests the notion of distribution of users over the results. Hence, for a given query, I assume that the user forms a probability distribution over the result set, which I refer as *market demand distribution*.

The above described distribution can also be derived from another perception. Assume that, the user prefers every search result with some extent. Then, a user's interest distribution over the search results can also be represented with a probability distribution. That means every user has his own distribution based on his preference over different search results. Then, for a given set of users in the market, I aggregate the individual distributions of all users which produces a consensuses distribution. This resulting distribution gives the other way of representing the market demand distribution over the considered result set across all the users.

In a nutshell, for a given result set and pool of users, the market demand distribution can be viewed as the average user interest distribution over the given result set. In the other approach, for a given specific result, its corresponding probability value in market demand distribution gives the number of users who prefers the result. However, in all of the experiments, I have considered the probability values of market demand distribution as the indication of number of users who prefers the result.

In a generic web search or e-commerce search platform, for an online search query, the size of relevant results is typically limited to a small number. This is evident from the fact that only a few users are willing to view/click the results presented after the first page. So, the user click activities after the first page are not useful to infer the implicit feedback. As the approach I put forward is based on the user click events in web search, it does make sense to restrict the relevant result set to the results in the first page. In a typical web or e-commerce search platform, the number of results shown in the first page is between 20-25. Hence, in the proposed simplified user behavior model, I assumed that the relevant result set has only 20 results.

3.1 Types of Market Demand Distribution

For the experimental purpose, I classify the search queries into different categories based on corresponding market demand distribution. Following are the types of distributions that I have considered.

Top3Biased Distribution: This distribution occurs for an easy query. For this type of query, the majority of the users are interested in a very small set of results. These results accommodate the substantial amount of user interest in the market. So, the market demand distribution is skewed towards these results. The above mentioned name "Top3Biased Distribution" refers to a distribution in which the significant amount of user interest is held by only top 3 results. The Figure 3.1 shows the histogram representation of a typical Top3Biased Distribution.

Example queries: Statue of liberty and Eiffel tower.

TwoCluster Distribution: This distribution occurs for an ambiguous query. For this type of query, the user interest can be clustered over a disjoint set of results. Each result set illustrates a specific meaning of that query. Figure 3.2 shows the histogram representation of a typical TwoCluster Distribution. In the figure, the results 1-10 belong to one cluster and the results 11-20 belong to another

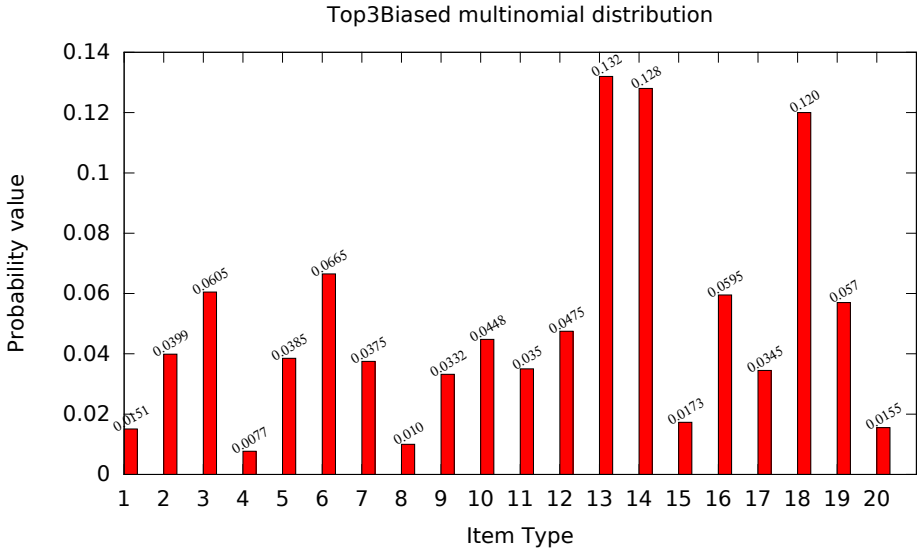

Figure 3.1.: Top3Biased Distribution

cluster.

Example queries: Java. For this query, the results 1-10 may corresponds to the Java island which is located at South-East Asia, On the other hand, the results 11-20 may refer to Java language which is a popular computer programming language.

NearUniform Distribution: This distribution occurs for a difficult query. For this type of query, the user interest is almost uniformly distributed over all the results. In this distribution, the probability values are closely-packed. Figure 3.3 shows the histogram representation of a typical NearUniform Distribution.

Example queries: The query iPhone in e-commerce search platform. This query is a difficult query since the same iPhone is available from many sellers. In this case, the ranking is too difficult as all the results represent the same product, and all results receive the same user interest level.

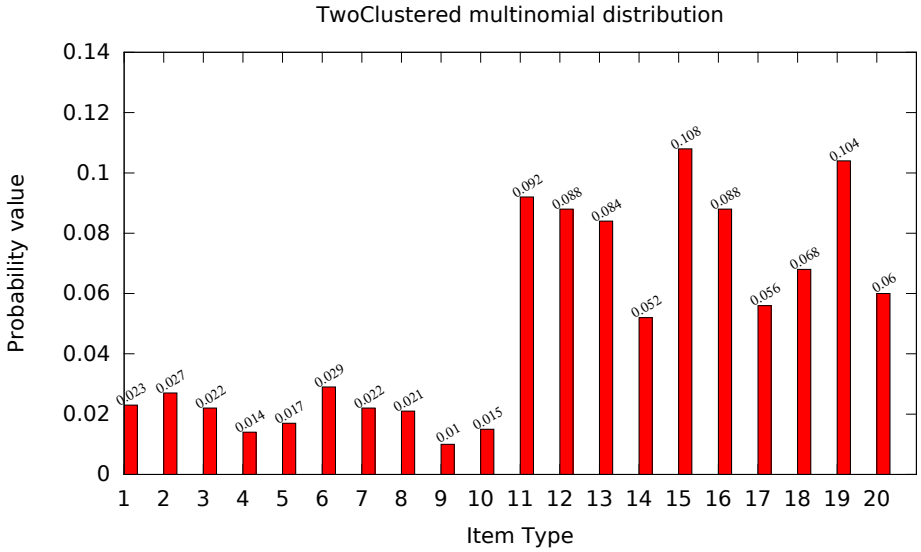

Figure 3.2.: TwoCluster Distribution

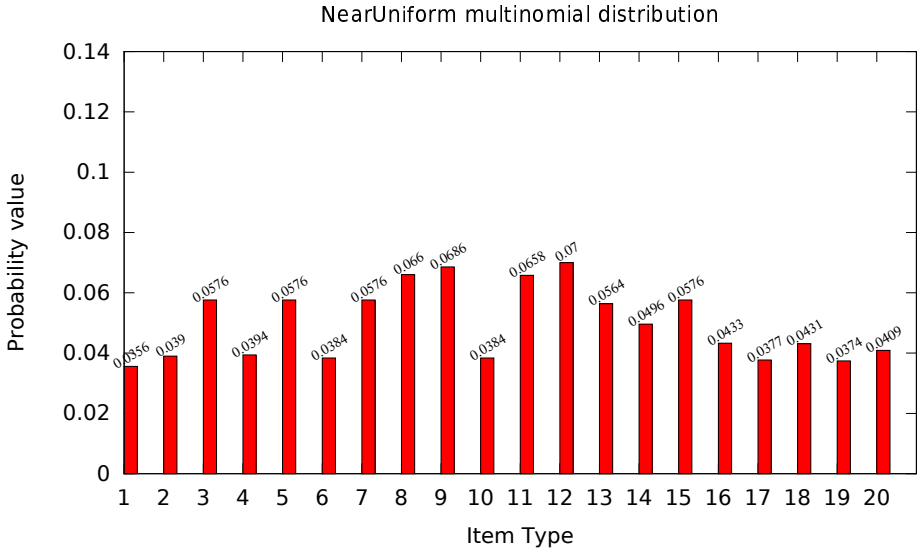

Figure 3.3.: NearUniform Distribution

3.2 User Behavior Under Color Ball Abstraction Model

I assume that the environment has infinite supply of balls of r distinct colors say blue, green, red, yellow etc. I also assume that a user is interested in buying only a

particular colored ball. I show the different colored balls in a sequence. This sequence is an analogy to the ranking task in an online search. If the user's interest matches with the color of the first ball, he buys it and stops his search endeavor. If the user is not interested in the current ball, he continues his search endeavor with an arbitrary but fixed probability. I refer this probability as *patience index* (p).

In the above Color Ball Abstraction Model, I introduce patience index to model the impact of position bias. Position bias is the phenomenon that a user is more likely to click a top ranked result than a lower ranked result even if the results are equally relevant. On the other hand, patience index denotes the fraction of users who examines the next ball if she is not interested in the current ball. Thus, patience index has a value between 0 and 1. Thus, the number of users who examines the next ball is lowered every time from the beginning of sequence to the end. If patience index is 0.6, it means that out of 100 only 60 times the user will look for the next ball. If the patience index is zero, the user will not continue his search if he does not find ball of his interest in the first location. Whereas if the patience index is 1.0, he will continue the search until he finds the ball of interest. I also assume that for each user the ball of interest is available somewhere in the list.

I define, the *efficiency* of a model as the ratio of total number of balls sold to the number of users visited. This gives a value between 0 and 1, which is the percentage of users that are satisfied with the retrieval system. The learning task that I am investigating is to discover the distribution of user interest which maximizes the efficiency of the model. Mathematically, the efficiency of a model can be represented as follows.

$$efficiency = \frac{\sum_{i=1}^{R} sold(i)}{nCustomers}$$

Where

R	represent total number of items considered.
$sold$	is the vector of size R that contains the number of sales for each item.
$nCustomers$	is the number of customers visited the model.

I first consider the obvious scenario, where no knowledge of user's preference over different colored balls is available. This assumption helps to find out the baseline efficiency of color ball abstraction model. As we don't have any prior knowledge, this baseline value of efficiency for this model can be calculated by uniform randomization of the balls. Since this assumption is the worst case scenario, the uniform randomization process gives a lower bound of the efficiency value for a general case. Following theorem provides a mathematical formulation for computing the efficiency value using the process of uniform randomization of balls.

Theorem 3.2.1 *For a Color ball abstraction model, if r is the number of different colors of the ball,* $Z = \{z_1, z_2, z_3, ..., z_r\}$ *is the market demand distribution over balls with different colors, and p is the patience index then the baseline efficiency value* $(eff_{baseline})$ *can be computed as* $\frac{1}{r(1-p)}$.

As the results are uniformly randomized, for a given user, the probability of occurrence of ball of his interest in each position is the same, which is equal to $\left(\frac{1}{r}\right)$.

$$\begin{aligned}
\text{The total probability of purchase for a user} &= \sum_{i=1}^{r} \frac{p^{i-1}}{r} \\
&= \frac{\sum_{i=1}^{r} p^{i-1}}{r} \\
&= \frac{1-p^r}{r(1-p)} \\
&= \frac{1}{r(1-p)} \quad (\text{As } p < 1,\ p^r \approx 0)
\end{aligned}$$

From the above expression, it is clear that for uniform random permutation, the probability of purchase for a given user is only dependent on the patience index (p)

and the total number of color of balls (r). Thus, this probability of purchase is identical for every user regardless of the color of ball he is interested in.

Hence, the expected number of customers that makes a purchase out of n customers

$$\begin{aligned} &= \frac{n}{r(1-p)} \\ eff &= \frac{n}{r(1-p)} \times \frac{1}{n} \quad \text{(by definition of the efficiency)} \\ &= \frac{1}{r(1-p)} \end{aligned}$$

At the other spectrum of the obvious scenario is total knowledge scenario, where the market demand distribution is completely known. In that case, the maximum efficiency can be achieved by ordering the balls in non-increasing order of their corresponding probability value. The following theorem proves this claim.

Theorem 3.2.2 *For a Color ball abstraction model, if r is the number of different colors of the ball, $Z = \{z_1, z_2, z_3, ..., z_r\}$ is the market demand distribution over balls with different colors, p is the patience index and $Z_s = \{z_{s1}, z_{s2}, z_{s3}, ..., z_{sr}\}$ is a sorted vector according to non-increasing order of Z. The maximum efficiency can be achieved by presenting the balls in an order R_s corresponding to Z_s.*

Let me take any arbitrary permutation of Z as $R_a = \{z_{a1}, z_{a2}, z_{a3}, ..., z_{ar}\}$. Then efficiency of the system using the permutation R_a would be

$$\begin{aligned} eff_{R_a} = (z_{a1}) + (z_{a2} \times p) + ... + (z_{am} \times p^{m-1}) + ... \\ +(z_{an} \times p^{n-1}) + ... + (z_{ar} \times p^{(r-1)}) \end{aligned}$$

Let me assume that R_a is not in non-increasing order. Then, there exist a pair of positions m and n in R_a such that $m < n$ and $z_{am} < z_{an}$. Now, I swap the balls at position m and n to create a new permutation $R_{a'}$. Then the efficiency of the system using the ranking $R_{a'}$ would be

$$\begin{aligned} eff_{R_{a'}} = (z_{a1}) + (z_{a2} \times p) + ... + (z_{an} \times p^{m-1}) + ... \\ +(z_{am} \times p^{n-1}) + ... + (z_{ar} \times p^{(r-1)}) \end{aligned}$$

Subtracting the efficiency of R_a from $R_{a'}$

$$\begin{aligned} eff_{R_{a'}} - eff_{R_a} &= (z_{an} \times p^{m-1}) + (z_{am} \times p^{n-1}) - (z_{am} \times p^{m-1}) - (z_{an} \times p^{n-1}) \\ &= (z_{an} \times p^{m-1}) - (z_{an} \times p^{n-1}) + (z_{am} \times p^{n-1}) - (z_{am} \times p^{m-1}) \\ &= z_{an} \times (p^{m-1} - p^{n-1}) - z_{am} \times (p^{m-1} - p^{n-1}) \\ &= (p^{m-1} - p^{n-1}) \times (z_{an} - z_{am}) \end{aligned}$$

As we already know $0 < p < 1$ and $m < n$, implies $(p^{m-1} - p^{n-1}) > 0$, from the above assumption $z_{am} < z_{an}$ implies $(z_{an} - z_{am}) > 0$.
Since both the terms are positive, I can conclude that

$$\begin{aligned} eff_{R_{a'}} - eff_{R_a} &= (p^{m-1} - p^{n-1}) \times (z_{an} - z_{am}) > 0 \\ eff_{R_{a'}} &> eff_{R_a} \end{aligned}$$

From the above expression, the efficiency of model can be improved by swapping the results z_{am} and z_{an}. I can repeatedly apply this to all $\binom{r}{2}$ pair of terms which in turn converts R_a into R_s. Thus, the permutation R_a has the best efficiency. So, the best efficiency of the model can be mathematically represented as follows.

$$eff_{max} = (z_{s1}) + (z_{s2} \times p) + ... + (z_{sm} \times p^{m-1}) + ... + (z_{sr} \times p^{(r-1)})$$

4 PRELIMINARIES

In this chapter, I introduce the definitions, notations and terminology which will be used throughout my thesis. In the typical web search activity, a user initializes search endeavor by submitting a *query* to search engine. The search engine evaluates the query and sends back the *results* in a specific ranking order. Here, I use the terms *document*, *items*, *balls* (in color ball abstraction model) and *results* interchangeably. I also refer the users as the *customers* in color ball abstraction model.

4.1 Definitions

Following are the terms and their definitions, which will be used in later chapters of my thesis.

Split position: This is the transformation point where the color ball abstraction model moves from exploration phase to exploitation phase. At this point the model stops the process of learning by randomizing the documents, but uses deterministic permutation over learned probabilities to maximize the revenue.

Deterministic Permutation: This permutation over a distribution (say Z) refers to arranging the items in non-increasing order of corresponding value in distribution Z. This permutation is fixed for a given distribution unless tie situation. Mathematically, this can be presented as follows.

Let the distribution and corresponding item set are $Z = \{z_1, z_2...z_r\}$ and $E = \{e_1, e_2...e_r\}$. If the distribution Z_{sort} is sorted vector according to non-increasing order of Z. Let $Z_{sort} = \{z_{s(1)}, z_{s(2)}...z_{s(r)}\}$, i.e. $z_{s(1)} >= z_{s(2)} >= ... >= z_{s(r-1)} >= z_{s(r)}$. The deterministic permutation over Z refers to the permutation $E_{determin} = \{e_{s(1)}, e_{s(2)}...e_{s(r)}\}$.

Probabilistic Permutation: The probabilistic permutation over a distribution (Z) refers to presentation order by probabilistically choosing every item from the distribution Z.

Let the distribution and corresponding item set are $Z = \{z_1, z_2...z_r\}$ and $E = \{e_1, e_2...e_r\}$. Then, the probabilistic permutation (E_{pp}) over Z is as follows.

$$E_{pp} = \{e_{pp(1)}, e_{pp(2)}...e_{pp(r)}\}$$

The probability an item to be in the first position of permutation is

$$\begin{aligned} Pr(e_{pp(1)} = e_1|Z) &= z_1 \\ Pr(e_{pp(1)} = e_2|Z) &= z_2 \\ &.... \\ Pr(e_{pp(1)} = e_r|Z) &= z_r \end{aligned}$$

The probability an item to be in the second position of permutation can be represented as follows.

Let $e_{pp(1)} = e_1$ and $Z' = \{z'_2, z'_3, z'_4...z'_r\}$ is the normalized distribution of Z after taking out element corresponding to the first position of permutation (z_1). i.e. Z' is the distribution of size $r - 1$.

$$\begin{aligned} Pr(e_{pp(2)} = e_2|e_{pp(1)} = e_1, Z) = z'_2 \\ Pr(e_{pp(2)} = e_3|e_{pp(1)} = e_1, Z) = z'_3 \\ ... \\ Pr(e_{pp(2)} = e_r|e_{pp(1)} = e_1, Z) = z'_r \end{aligned}$$

Similarly, the probability an item to be in the third position of permutation can be represented as follows.

Let $e_{pp(1)} = e_1$, $e_{pp(2)} = e_2$ and $Z'' = \{z''_3, z''_4 ... z''_r\}$ is the normalized distribution of Z after taking out elements corresponding to the first and second positions of permutation (z_1 and z_2). i.e Z'' is the distribution of size $r - 2$.

$$Pr(e_{pp(3)} = e_3 | e_{pp(1)} = e_1, e_{pp(2)} = e_2, Z) = z''_3$$
$$Pr(e_{pp(3)} = e_4 | e_{pp(1)} = e_1, e_{pp(2)} = e_2, Z) = z''_4$$
$$...$$
$$Pr(e_{pp(3)} = e_r | e_{pp(1)} = e_1, e_{pp(2)} = e_2, Z) = z''_r$$

As we chooses the items probabilistically, the final outcome of each try (presentation order) may not be identical with the subsequent trails. So, unlike the deterministic permutation approach the presentation is not unique in this approach. There is always chance for exploration in this permutation. But, because of characteristic of probabilistic selection of the items, most of the times we can expect the outcome to be close to the deterministic permutation.

4.2 Performance Evaluation Metrics

To evaluate the performance of the proposed algorithm, I used standard metrics like precision at n ($P@n$), Discounted Cumulative Gain (DCG), Normalized Discounted Cumulative Gain ($NDCG$) and Kendall tau rank correlation. These metrics are extensively used for comparing the rankings in information retrieval domain.

Precision at n ($P@n$) [34]: This metric denotes the relevance of first n documents of a given query. For example, if the relevance of the first 10 documents returned for a query are $\{1, 1, 0, 1, 0, 0, 1, 0, 1, 0\}$, where 0 denotes non-relevant and 1 denotes relevant document. Then precision at 1 to precision at 10 would be

$$\left\{\frac{1}{1}, \frac{2}{2}, \frac{2}{3}, \frac{3}{4}, \frac{3}{5}, \frac{3}{6}, \frac{4}{7}, \frac{4}{8}, \frac{5}{9}, \frac{5}{10}\right\}.$$

For a given query the precision at n ($P@n$) metric is defined as follows.

$$P@n = \frac{\sum_{i=1}^{n} r(i)}{n}$$

where $r(i)$ denotes the relevance of i^{th} document in ranking list.

Discount Cumulative Gain (DCG) [35]: For any given ranking list of size n, the Discount Cumulative Gain can be calculated as follows.

$$DCG = \sum_{i=1}^{n} \frac{2^{r(i)} - 1}{log(i+1)}$$

where $r(i)$ denotes the relevance of i^{th} document of ranking list. The numerator term $2^{r(i)} - 1$ denotes the gain for the i^{th} document of ranking list, where the term $\frac{2^{r(i)}-1}{log(i+1)}$ denotes the discounted gain. So the term $\sum_{i=1}^{n} \frac{2^{r(i)}-1}{log(i+1)}$ gives the cumulative gain over all n documents.

Normalized Discount Cumulative Gain ($NDCG$) [35]: This metric gives the ratio of the DCG value of current ranking to the maximum DCG. The maximum DCG could be found for the perfect ranking. The $NDCG$ can be calculated by multiplying the DCG with constant Z_n, which is a normalization constant. This should be chosen in such a way that perfect ranking gets the $NDCG$ value as 1. Following is the mathematical representation of $NDCG$ calculation to the given ranking of the size n.

$$NDCG = Z_n \times \sum_{i=1}^{n} \frac{2^{r(i)} - 1}{log(i+1)}$$

where $r(i)$ denotes the relevance of i^{th} document in ranking list.

Kendall tau Rank Correlation: [36] This metric assess the degree of association between a pair of ranking orders. This rank correlation coefficient relies upon

the number of inversions of pairs of documents which would be needed to reorganize an order to reach the other order. Before giving mathematical definition of Kendall tau rank correlation, let me define terms *concordant pairs* and *discordant pairs* which will be useful in calculating this metric.

Let $A = (a_1, a_2..., a_n)$ and $B = (b_1, b_2, ..., b_n)$ be any two ranking permutations. Any pair of documents at position i and j i.e. (a_i, a_j) and (b_i, b_j) could be considered as concordant pairs if both ranks agree with the document's order. That means, either $a_i > a_j$ and $b_i > b_j$ or $a_i < a_j$ and $b_i < b_j$. Similarly, they could be considered as discordant pairs if both ranks disagree with the document's order, that means, either $a_i > a_j$ and $b_i < b_j$ or $a_i < a_j$ and $b_i > b_j$.

Mathematically, the Kendall tau rank correlation is defined as follows.

$$\tau = \frac{\mathcal{CP} - \mathcal{DP}}{\mathcal{CP} + \mathcal{DP}}$$

Where, $\mathcal{CP}$ denotes the count of concordant pairs and $\mathcal{DP}$ denotes the count of discordant pairs. τ = 1 represents that both rankings are absolutely the same. On the other hand, τ = -1 represents that the rankings are completely contradictory. This value vary between $[-1, 1]$.

5 BACKGROUND

In this chapter, I would be introducing concept of learning automata which I have used in my proposed algorithm.

5.1 Learning Automata

An automata can be considered as a learning agent (automaton) operating in an abstract random environment. The automaton has choice among a finite set of allowable actions. We consider that these actions are performed recursively in the random environment. In each decision making process, the automaton chooses an action from those set of available actions. The random environment evaluates the action and gives a response among a set of allowable outputs. These outputs are probabilistically connected to the action chosen. This response from the environment will be used by the automaton for finding future actions. The idea of learning automata is to regulate how the selection of action at every stage should be guided according to past actions and responses. By repeating this process the agent learns to select the action having the best reward. Automaton uses a learning algorithm to find the next action from responses of the past actions. The key insight here is to ensure that the decision will be made with minimum information about the environment.

In a nutshell, the learning automaton can be considered as an automaton that improves its performance by performing actions in an abstract random environment. The automaton objective is to find the optimal action. The optimal action has the highest probability of producing the favorable output. This optimal action is unknown to the learning agent (automaton). The agent explores over all actions and uses the response of the environment to identify the optimal action. This process of automaton

acting in unknown environment to improve its performance in specific fashion is referred as learning automata (LA).

In the following sections, I provide precise description of the entities that an learning automaton consists of.

5.1.1 Environment

The term *environment* can be considered as a "large class of general unknown media in which an automaton or group of automaton can operate" [37]. Mathematically, a binary environment (P-Model) (Figure:5.1) can be defined as the following triplet.

$$\Big\{\alpha, c, \beta\Big\}$$

where $\alpha = \{\alpha_1, \alpha_2, \alpha_3 ... \alpha_r\}$ is a finite action set, $\beta = \{\beta_1, \beta_2\}$ is the environment response set and $c = \{c_1, c_2 ... c_r\}$ is a set of penalty probabilities, where each c_i corresponds to one action α_i.

A specific action $\alpha(n)$ belongs to the set α is the input to the environment which can be chosen at discrete time $t = n$ $(0, 1, 2...)$. The response of the environment for the action $\alpha(n)$ is referred as $\beta(n)$, which can take either β_1 or β_2 in a binary environment. For mathematical advantage, these two values are represented as 0 and 1. An output $\beta(n) = 1$ is considered as unfavorable response or penalty or failure, whereas $\beta(n) = 0$ is considered as favorable response or reward or success. The element c_i refers to the probability of getting unfavorable response for the specific action α_i. This value represents the behavior of environment for the action α_i. Mathematically, the element c_i can be represented as follows.

$$Pr(\beta(n) = 1 | \alpha(n) = \alpha_i) = c_i, \qquad \text{where} \quad i = (0, 1, 2...r)$$

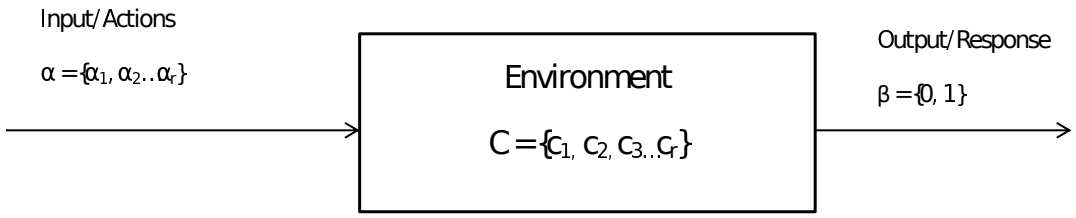

Figure 5.1.: Binary Environment (Ref: Learning Automata an Introduction)

5.1.2 Automaton

The *automaton* can be considered as the set of abstract systems that interact with the random environment. Primarily, the automaton outputs the series of action by taking the series of responses. Mathematically, an automaton (Figure:5.2) can be represented with a quintuple.

$$\Big\{\phi, \alpha, \beta, F(.,.), G(.,.)\Big\}.$$

Where the term ϕ denotes the internal states of the automaton. At any given instant n the state of an automaton can be referred as $\phi(n)$, which belongs to the finite set ϕ.

$$\phi = \{\phi_1, \phi_2 ... \phi_s\},$$

The output of automaton (action to be chosen) at instant n is $\alpha(n)$, which belongs to the finite set α,

$$\alpha = \{\alpha_1, \alpha_2 ... \alpha_r\}.$$

The input of automaton at instant n is $\beta(n)$, which belongs to the finite set β.

$$\beta = \{\beta_1, \beta_2\} \; or \; \{0, 1\} \qquad \text{for a binary environment.}$$

The *transition function* F regulates the state at the current instant $(n + 1)$ w.r.t previous state and response of the environment.

$$\phi(n+1) = F\Big[\phi(n), \beta(n)\Big],$$

The *output function* G identifies the output of the automaton at the instant n w.r.t to the current state.

$$\alpha(n) = G\Big(\phi(n)\Big).$$

Based on the characteristics of the transition function (F) and output function (G), the automaton can be classified as deterministic automaton and stochastic automation.

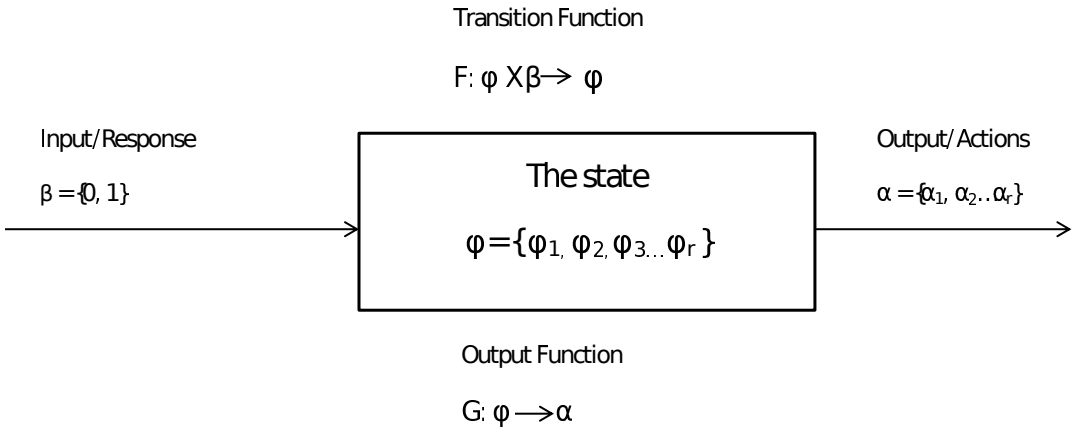

Figure 5.2.: Automaton (Ref: Learning Automata an Introduction)

5.1.3 Deterministic Automaton

In this type of automaton, the transition function (F) and output function (G) are deterministic mappings. In this case, for a specific initial state and response of the environment the actions to be chosen are predefined. That means for every instant that we come across same combination of state and environment response the output action is the same. In other words, if the input set (environment responses) is finite, the transition function (F) and output function (G) can be simply represented as a matrix or a graph.

5.1.4 Stochastic Automaton

In stochastic automaton, at least one of these functions F and G is stochastic. The outcome (actions to be chosen) of the automaton varies with the instant (n) even-though the input and state of automaton is identical.

If the transition function F is stochastic for the specific state (ϕ_i) and input (β), the next state is not unique. Therefore, the function F gives the probability distribution of reaching those states. F can be represented as the set of conditional probability matrices $F(\beta_1), F(\beta_2)..F(\beta_m)$ (for binary m=2), where each matrices is of the size $s \times s$, whose entries f_{ij}^{β} can be represented as follows.

$$\begin{aligned} f_{ij}^{\beta} = Pr\{\phi(n+1) = \phi_j | \phi(n) &= \phi_i, \beta(n) = \beta\} \\ i &= 1, 2, 3 \ldots s \\ j &= 1, 2, 3 \ldots s \\ \beta &= \beta_1, \beta_2 \ldots \beta_m \end{aligned}$$

Likewise, if G is stochastic, G can be represented by set of conditional probability matrices of size $s \times r$. The entries of g_{ij} can be represented as follows.

$$\begin{aligned} g_{ij} = Pr\{\alpha(n) = \alpha_j | \phi(n) &= \phi_i\} \\ i &= 1, 2, 3 \ldots s \\ j &= 1, 2, 3 \ldots r \end{aligned}$$

The stochastic automaton can be further divided into fixed structure and variable structure automata.

Fixed structure stochastic automata: In the *fixed structure automaton* the stochastic transition functions (F and G) have a fixed probability distribution. These functions have specific fixed distribution regardless of the time (n) and input.

Variable structure stochastic automata: The basic operation performed by a *Variable structure stochastic automata* is the updating of action probabilities with respect to response of the environment. Greater flexibility can be achieved in stochastic learning automata models by modeling general stochastic system by which the state transition probabilities can be updated with the time using a reinforcement scheme.

5.2 Behavior of Learning Automata

The fundamental objective of learning automata is to justify the intuitive notion of learning. Norm of behavior are the features of learning automata that are necessary to evaluate the learning process. For a specific action probability vector, *Average penalty* ($M(n)$) is one such feature which plays a key role for the comparison of different automata. For example, at some point of learning process, if the action α_i is chosen with probability $p_i(n)$ then average penalty given $p(n)$ can be calculated as follows.

$$\begin{aligned} M(n) &= E\Big[\beta(n)|p(n)\Big] \\ &= Pr\Big[\beta(n) = 1|p(n)\Big] \\ &= \left[\sum_{i=1}^{r} p_i(n)c_i\right] \end{aligned}$$

Expedient: Learning automata is said to be *expedient* when the learning automation average penalty is less than choosing actions absolute random manner. In this case it is only guaranteed that learning automata can only perform better than pure random selection. Which means it may not be necessarily optimized.

$$\lim_{n\to\infty} E\Big[M(n)\Big] < M_0$$

Where M_0 is the average penalty for an absolute random strategy.

Optimal: To improve the performance of learning automata, it is advisable to choose the selection of actions such a way that average penalty can be minimized. In this case, the automata is known to be *optimal*. The optimality ensure that the automata asymptotically chooses the action having the least penalty with probability one. The optimality can be represented as follows.

$$\lim_{n\to\infty} E\Big[M(n)\Big] = c^*$$

Where c^* is the minimum value of the penalty set i.e. $\{c_1, c_2...c_r\}$

ϵ-optimal: Achieving the optimality in a real-life situation is not generally possible. In this case sub-optimal performance *ϵ-optimal* might comes in handy. This means the average penalty of learning automata gets close to the least value. We can make it as close as we want. This behavior can be mathematically represented as

$$\lim_{n\to\infty} E\Big[M(n)\Big] < c^* + \epsilon$$

Absolutely Expedient: The feature called *absolutely expedient* is useful in judging the behavior of learning automaton. Learning automaton is said to be *absolutely expedient* if average penalty is absolutely monotonically decreasing at every instant (n) of iteration process.

$$E\Big[M(n+1)|P(n)\Big] < E\Big[M(n)\Big]$$

5.3 Variable Structure Stochastic Automata

As described in the previous section, the variable structure stochastic automata, modifies the action probabilities or transition probabilities. Mathematically, the variable structure stochastic automata can be represented as follows.

$$\{\phi, \alpha, \beta, A(.,.), G(.,.)\},$$

Here, A is a general updating scheme for action probability vector (P), where as the other symbols holds the definition given in previous section. A can also be referred as reinforcement scheme.

5.3.1 Reinforcement Schemes

To justify the notion of learning in automata, it is necessary to update the action probability vector (P) in every step. This learning algorithm will be referred as Reinforcement Scheme. Because of this it is evident that reinforcement scheme has vital role in the performance of a learning automata. Reinforcement scheme can be commonly described as follows

$$p(n+1) = T\Big[p(n), \alpha(n), \beta(n)\Big]$$

where T is an operator and $P(n)$ is the action probability vector at an instant t.

Reinforcement schemes can be classified either on the basis of behavior exhibited by the learning automaton (optimal or expedient) or on the character of function (T) used in scheme (non-liner, linear and mixed). Each of them have various types of convergence of action probabilities and different rates of convergence. The detailed discussion about the convergence of learning automaton can be found in subsequent sections.

The key idea in the reinforcement scheme is quite straightforward. For any action α_i chosen by learning automation, if a penalty input occurs then probability of $p_i(n)$ will be decreased and all other elements probability is increased by the same factor. But, sometimes the transition probabilities may be unaltered even for penalty which is known as *inaction*. For a non-penalty input then probability of $p_i(n)$ will be increased and all other actions probabilities will be fairly decreased.

5.3.2 Asymptotic Behavior of Variable Structure Stochastic Automata

For the variable structure stochastic automata, the asymptotic behavior of learning automata is purely dependent on the characteristics of the reinforcement model. The convergence of ergodic schemes are completely different form the absolutely expedient schemes.

Absolutely Expedient Schemes: For the learning automata using absolutely expedient schemes the action probability vector converges to one of the action with probability close to 1, which are known as absorbing states or absorbing barriers. The initial value of probability vector $p(0)$ plays the key role in selection of the absorbing state by the learning automata. The linear reward and inaction schemes (L_{R-I}) are the best example for these schemes.

Since this automata chooses only a single action with probability close to 1 and remaining all actions receives the negligible value, this model is not suitable for ranking problems. If we can generate the relative preference among the actions that could be ideal for solving the ranking problems.

Ergodic Schemes: The characteristics of learning automata based on ergodic schemes are totally different with absolute expedient schemes. This model exhibits different mode of convergence than previous scheme. This scheme exhibits the distance diminishing property. This is the reason for ergodic behavior of the scheme. The action probability vector $p(n)$ converges to random vector p^* whose values are independent of initial value of action probability vector $p(n)$.

The example for these schemes are linear reward and penalty schemes (L_{R-P}), provided if the environment is stationary and it doesn't have a pure optimal or negative ($c_i = 0$ or 1) action. For L_{R-P} scheme the action probability vector $p(n)$ converges to p^* whose values are inversely proportionate to average loses. This scheme stabilizes $p(n)$ at this equilibrium point.

Hence, these schemes stabilizes $p(n)$ at the equilibrium point, it makes these model highly suitable for solving the ranking problems. We can use the individ-

ual action probability values of $p_i^* \epsilon\ p^*(n)$ can be used as the relative preference judgments of the corresponding action. In the simulation I have implemented Bush-Mosteller scheme (L_{R-P}) to solve the ranking problem.

5.3.3 Bush-Mosteller Scheme

This is one of the commonly used linear reward and penalty reinforcement scheme. This model was proposed by the Bush and Mosteller. This scheme can be represented in the following vector form.

$$p(n+1) = p(n) + \gamma_n \times \left[e(x_n) - p_n + \beta_n \frac{\left(e^R - R \times e(x_n)\right)}{(R-1)} \right]$$

where

$p_i(n) > 0 \quad (i = 1, 2 \ldots R)$,

γ_n(Correction factor) $\quad \epsilon \quad (0, 1)$,

$\beta_n$is response of the environment at time n,

$e^R = (1, 1, \ldots 1)^T$ Vector of R dimensions having all values as 1 ,

$e(x_n) = (0, \ldots 0, 1, 0 \ldots 0)^T$ Vector with n^{th} component as 1 and others as 0,

$p(n) = (p_1(n), p_2(n) \ldots p_R(n))^T$Action probability vector at instant n.

When the average loss function corresponding to optimal action (c^*) is equal to zero and correction factor is constant ($\gamma_n = \gamma$) then this scheme converges action probability vector in the direction of optimal solution. In other words the action probability value corresponding to the optimal action reaches close to 1 and remaining action probabilities become negligible. But, this type of convergence is not useful to solve ranking problem. In fact, the ranking problem can be solved if we can identify the relative preferences. These relative preferences can be calculated in the other type convergence property exhibited by Bush-Mosteller scheme.

If the environment doesn't have pure optimal action (average loss of optimal action is zero) and correction factor tends to zero, this scheme leads to different mode of convergence. This scheme converts the probability vector to a vector whose values are inversely proportional to the average losses of corresponding actions [38]. i.e.,

$$p_i(n) \rightarrow \frac{K}{c_i}$$

When

$$c^* \neq 0,$$
$$\sum_{t=1}^{\infty} \gamma_n^2 < \infty$$

Where

$p(n)$	is the action probability vector at time n
K	is a constant
c_i	is average penalty of action i
c^*	is average penalty of optimal action*and*
γ_n	is correction factor at time n

The above Bush-Mosteller scheme can be represented in more elaborated form if it operates in a binary environment. As the binary environment only generates binary response (0 or 1), the above scheme can be represented as the following split function.

if $\gamma = 1$ (penalty)

$$\begin{aligned} p_i(n+1) &= p_i(n) - \gamma_n[p_i(n)], \quad \text{where} \quad \alpha(n) = \alpha_i \\ p_i(n+1) &= p_i(n) + \gamma_n\Big[p_i(n) - \frac{1}{R}\Big], \quad \text{where} \quad \alpha(n) \neq \alpha_i \end{aligned}$$

if $\gamma = 0$ (reward)

$$\begin{aligned} p_i(n+1) &= p_i(n) + \gamma_n[1 - p_i(n)], \quad \text{where} \quad \alpha(n) = \alpha_i \\ p_i(n+1) &= p_i(n) - \gamma_n[p_i(n)], \quad \text{where} \quad \alpha(n) \neq \alpha_i \end{aligned}$$

6 METHODS

6.1 Objective

The objective of my thesis is to optimize the search engine by resolving the imperfect orderings in search engine ranking. The ideal ranking is the ordering of the results in such a way that their corresponding values in *Market Demand Distribution* gives a non-increasing order. As I have already mentioned, the current model begins at a point where we have no information about *Market Demand Distribution*. So, the following models are the on-line approaches which converges the any given ordering to the ideal ordering using user click activities.

In the beginning, the lack of knowledge of user's interest suggests us to include the *exploration* task. But later we have to leverage the knowledge gained through this exploration task into ranking with the goal of maximizing sales, this process is known as *exploitation*. So, all of the models that I am going to discuss now consists of these two phases either implicitly or explicitly.

In this thesis, I implemented the following learning models.

1. Split Model
2. Overlapping Models
3. Learning Automata Based Ranking Model

6.2 Split Model

In this approach, the total process is divided into two phases. First part of the process is exploration during which we would be learning the *Market Demand Distribution*.

In this phase, the items will be shown to the user in a uniformly randomized order. Which means, every specific item has equal chance to be appeared in every position. Later in the second phase, to maximize the revenue we use the deterministic permutation using the information gained in exploration phase. This is the phase of exploitation.

The figure (Ref 6.1) presents the pseudo code for split model. The split model success is strongly based on the point where we move from exploration to exploitation. The key aspect in this approach is to identify the best split point. We must be careful while choosing the split position. If we choose the split position too early then the information gained may not be true. This leads to bad ranking during the exploitation phase. On the other hand, if we spend too much time on the exploration, we might get substantial knowledge but we couldn't leverage learned knowledge into our action as the remaining exploitation phase is not long.

6.3 Overlapping Models

In these models, we overlap the exploration and exploitation throughout the process. We further implemented these in 2 distinct approaches. The basic difference between those two approaches is the way how exploration and exploitation phases are integrated together.

6.3.1 Reinforcement Learning Model

In this model, we performs both exploration and exploitation phases together. The presentation of the documents in each phase is exactly similar to the split model. i.e. in exploration phase we use uniform randomization and in exploitation phase we use deterministic permutation using the current knowledge.

We defined an exploration factor *alpha.* This refers to the amount of time we have spent on exploration. This factor helps to choose the action between the exploration and exploitation. Alpha value is always between 0-1. This factor provides the portion

of time that we spent on the exploration. The rest of the time we used exploitation. In the simulation, I have implemented this model considering various values of alpha.

6.3.2 No-regret Learning Model

Unlike the split and reinforcement learning model, this doesn't have any separate exploration and exploitation phase. This model implicitly has the exploration and exploitation.

This model differs with reinforcement learning model in two aspects. First, it never uses deterministic permutation. It uses the probabilistic permutation based on a specific probability vector ($pVector$). There is always scope for the exploration in the probabilistic permutation. In the beginning, as we don't have any knowledge about customer distribution we choose this to be vector having same value for item types.

The second difference is the exploitation process. This process was incorporated in this process by updating $pVector$. The $pVector$ will be updated every-time a new sale is occurred. The $pVector$ is added with the sales vector to move the $pVector$ towards the sales. This process ensures $pVector$ slowly deviated in the direction of the sales count and in the end this will converge in the direction of sales vector. Every-time we normalize the $pVector$ to ensure the sum of all values is equals to 1.

6.4 Learning Automata Based Ranking Model

In this model, we leverage each and every action of the user into feedback. This model overcomes the limitation of not using the negative actions of user. Similar to no-regret learning model, the current model also has the $pVector$ which get updated every-time user either purchase the current item or move to next item. If the user purchase an item then the $pVector$ will be updated in such a way that the current item's probability value increases and correspondingly other item's probability will be

decreased. This update process is based on the Bush-Mosteller scheme. This model works for both positive (reward) and negative (penalty) scenarios.

To maintain the exploration process we choose the very first item probabilistically from the $pVector$. To maintain the exploitation process active, the other items (from second position to the last) of permutation are chosen deterministically in non-increasing order of $pVector$ probability value. During the user search endeavor we update the $pVector$ accordingly.

```
SPLIT MODEL (Z, nCustomers, initialPermutation, splitPosition, patienceIndex)
    Z: Market demand distribution
    nCustomers: No of Customers to be generated
    patienceIndex :  Patience factor of the user
    initialPermutation: The given initial permutation
 1  iter = 0
 2  efficiency = 0
 3  nRejections = 0
 4  salesVector =< 0, 0, 0....0 >
 5  while iter ≤ nCustomers
 6      i = 0
 7      if iter ≤ splitPosition
 8          presentation = uniform random permutation of initialPermutation
 9      else
10          learned probability = normalize salesVector
11          presentation = deterministic permutation of learned probability
12      while i is not at the end of presentation
13          if customer interest == presentation(i)
14              add 1 to corresponding value of sales vector
15              break
16          else
17              randomPatience = random value between 0 and 1
18              if randomPatience ≤ patienceIndex
19                  i = i + 1
20                  continue
21              else
22                  nRejections = nRejections + 1
23                  break
24      iter = iter + 1
25  efficiency = (nCustomers − nRejections) / nCustomers
26  return efficiency
```

Figure 6.1.: Split Model Pseudocode

```
REINFORCEMENT LEARNING MODEL(Z, nCustomers, initialPermutation,
alpha, patienceIndex)
    Z: Market demand distribution
    nCustomers: No of Customers to be generated
    initialPermutation: The given initial permutation
    patienceIndex : Patience factor of the user
    alpha: exploration factor
 1  iter = 0
 2  efficiency = 0
 3  nRejections = 0
 4  salesVector =< 0, 0, 0....0 >
 5  while iter ≤ nCustomers
 6      i = 0
 7      random = random value between 0 and 1
 8      if random ≤ alpha
 9          presentation = uniform random permutation of initialPermutation
10      else
11          learned probability = normalize salesVector
12          presentation = deterministic permutation of learned probability
13      while i is not at the end of presentation
14          if customer interest == presentation(i)
15              add 1 to corresponding value of sales vector
16              break
17          else
18              randomPatience = random value between 0 and 1
19              if randomPatience ≤ patienceIndex
20                  i = i + 1
21                  continue
22              else
23                  nRejections = nRejections + 1
24                  break
25      iter = iter + 1
26  efficiency = (nCustomers − nRejections) / nCustomers
27  return efficiency
```

Figure 6.2.: Reinforcement Learning Model Pseudocode

```
No-Regret Learning Model(Z, nCustomers, initialPermutation,
patienceIndex)
    Z: Market demand distribution
    nCustomers: No of Customers to be generated
    initialPermutation: The given initial permutation
    patienceIndex : Patience factor of the user
 1  iter = 0
 2  efficiency = 0
 3  nRejections = 0
 4  R = initialPermutation size
 5  while iter ≤ nCustomers
 6      i = 0
 7      pVector =< 1/R, 1/R, ....1/R >
 8      presentation = probabilistic permutation of pVector
 9      while i is not at the end of presentation
10          if customer interest == presentation(i)
11              add 1 to corresponding value of sales vector
12              learned probability = normalize the salesVector
13              pVector = pVector+ learned probability Vector
14              normalize the pVector
15              break
16          else
17              randomPatience = random value between 0 and 1
18              if randomPatience ≤ patienceIndex
19                  i = i + 1
20                  continue
21              else
22                  nRejections = nRejections + 1
23                  break
24      iter = iter + 1
25  efficiency = (nCustomers − nRejection) / nCustomers
26  return efficiency
```

Figure 6.3.: No-Regret Learning Model Pseudocode

```
LEARNING AUTOMATA BASED LEARNING MODEL(Z, nCustomers,
initialPermutation, patienceIndex)
    Z: Market demand distribution
    nCustomers: No of Customers to be generated
    initialPermutation: The given initial permutation
    patienceIndex : Patience factor of the user
 1  iter = 0 ; nRejections = 0
 2  R = initialPermutation size
 3  pVector =< 1/R, 1/R, ....1/R >; salesVector =< 0, 0, 0....0 >
 4  while iter ≤ nCustomers
 5      i = 0 and γ = 0
 6      presentation(0) = probabilistically choose item from pVector
 7      Fill presentation(1, 2, ...R − 1) with deterministic permutation of
        pVector after removing item chosen at first position of presentation
 8      while i is not at the end of presentation
 9          γ = 1/(iter+1)
10          if customer interest == presentation(i)
11              add 1 to corresponding value of sales vector
12              β =0 (reward action)
13              pVector = BushMosteller(pVector,presentation(i),β,γ)
14              break
15          else
16              randomPatience = random value between 0 and 1
17              if randomPatience ≤ patienceIndex
18                  β =1 (penalty action)
19                  pVector =BushMosteller(pVector, presentation(i),β,γ)
20                  i = i + 1; continue
21              else
22                  nRejections = nRejections + 1; break
23      iter = iter + 1
24 finalPermutation = deterministicPermutation of pVector
25 idealPermutation = deterministicPermutation of Z
26 Find kendall tau correlation, Precision@10, DCG and NDCG
```

Figure 6.4.: Learning Automata Based Learning Model Pseudocode

BushMosteller($pVector, i, \beta, \gamma$)

$pVector$: Probability distribution

i: Action chosen

β: Environment Reaction

γ: Update Step size

e^R: $(1, 1, ..1)^T$ Vector of R dimensions having all values as 1

$e(x_i)$: $(0, ...0, 1, 0...0)^T$ Vector with i^{th} component as 1 and others as 0

1 $pVector = pVector + \gamma_n \times [e(x_i) - p_n + \beta_n \frac{(e^R - R \times e(x_i))}{(R-1)}]$

2 **return** $pVector$

Figure 6.5.: BushMostellerScheme Pseudocode

7 EXPERIMENTS AND RESULTS

I have implemented all models as a Java application. I ran all the experiments in a computer having CPU @ 2.20GHz processor and 6GB RAM using the Linux operating system.

7.1 Experiments

Here, the objective of my experiments is to identify the best ranking based on hidden user distribution of market demand. The information gained on user distribution could be leveraged as document ranking.

7.1.1 Split Model

The key aspect in this approach is to identify the proper point of transition (split position) from exploration to exploitation. We must be careful while choosing this split position. If the split position too early then the information gained may not be true. This leads to get a bad ranking during the exploitation phase. On the other hand, if the split position is too far, then model spent longer on exploration, we might get substantial knowledge but we may loose customer trust because of the prolonged exploration phase.

Experiment:

Following is the information that I have used in my split model simulation.

User Distributions considered are top3Biased distribution (Figure: 3.1), twoCluster distribution (Figure: 3.2) and nearUniform distribution (Figure: 3.3). The patience index considered for all these three distributions is 0.35.

Total number of customers generated are 5×10^3. The Figures 7.1, 7.2 and 7.3 represent plots between the split position and efficiency for the above mentioned distributions respectively.

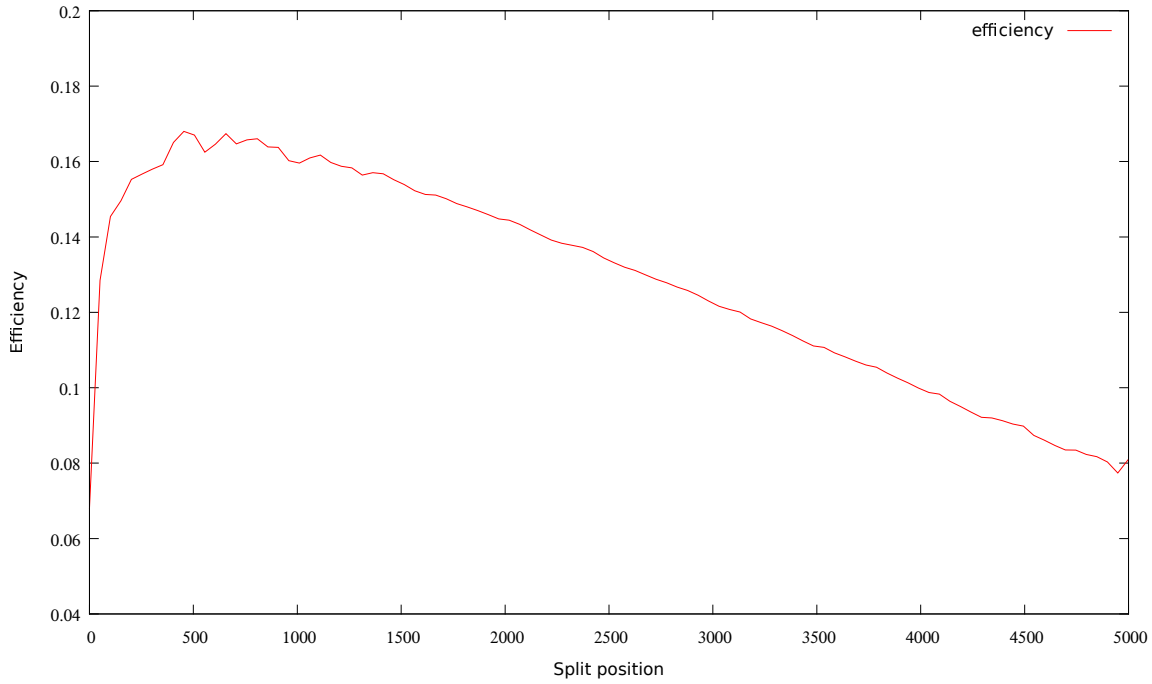

Figure 7.1.: Split vs Efficiency Plot for Top3Biased Distribution

Observations:

All of these plots exhibits similar behavior. If the split position is small the efficiency is too low. This is because the model hasn't spend sufficient amount of time for learning the user distribution. The efficiency value increases till some point and then changes its behavior and start descending. There is a change in the trend of efficiency because as we are moving split point to the right, we will be spending extra time on exploration and unable to leverage the gained knowledge in ranking. So it is always better to stop the exploration process as soon as we gain sufficient knowledge of the distribution. But unfortunately we can't derive any mathematical equation to relate the best split position with distribution. In this model, the exploration and exploitation phases are strictly separated and hence the identification of ideal split

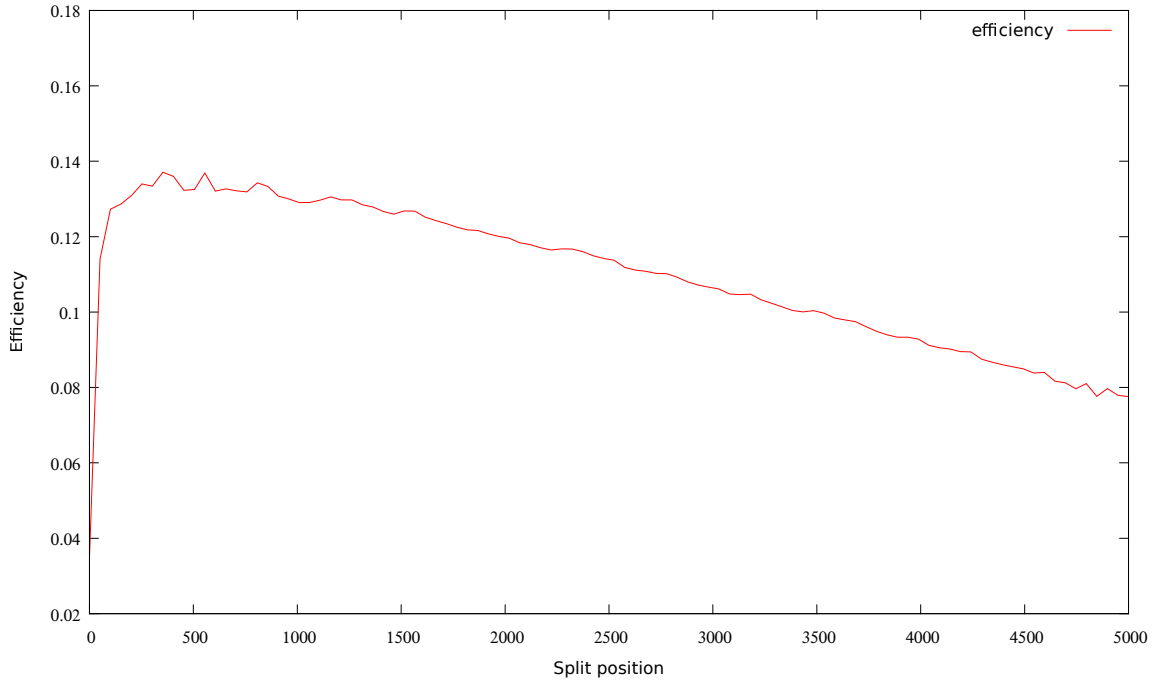

Figure 7.2.: Split vs Efficiency Plot for TwoCluster Distribution

point is a compulsion. This limitation has lead us to build the next models which doesn't have the strict separation point between the exploration and exploitation process.

7.1.2 Overlapping Models

As the name suggests, the models I am going to discuss have overlapping of the exploration and exploitation phase. The major difference between the models mentioned below is the way how exploration and exploitation phases were combined together. In reinforcement learning model we can explicitly identify the phases, but in No-regret learning model these two processes are completely blent.

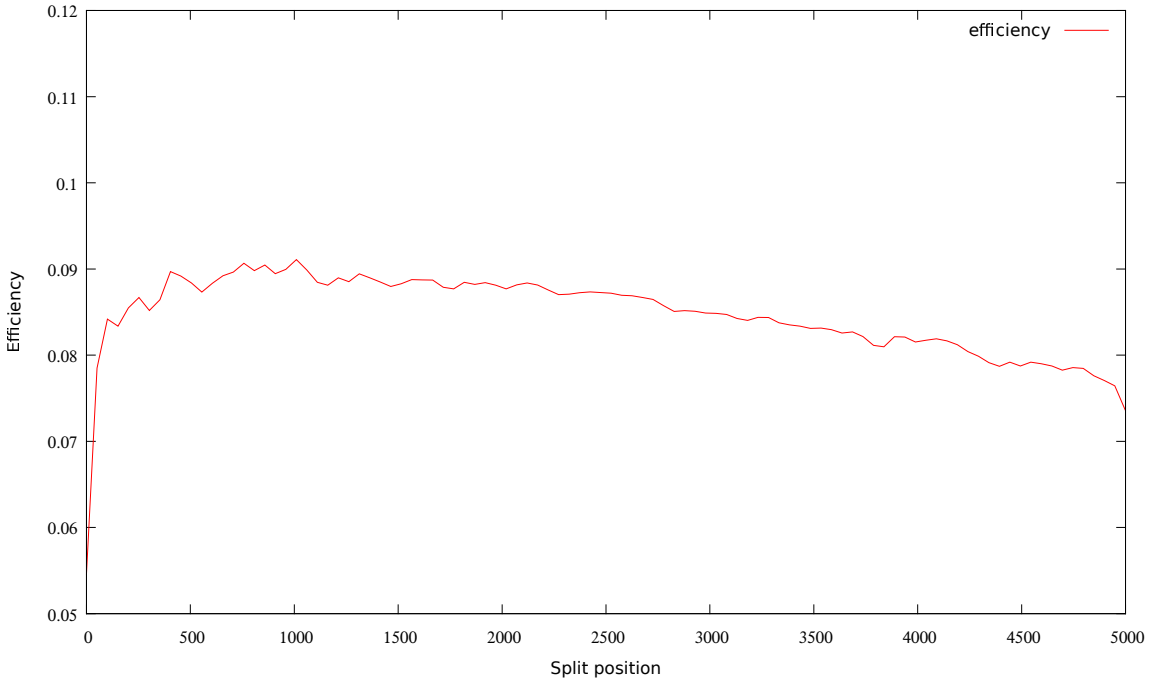

Figure 7.3.: Split vs Efficiency Plot for NearUniform Distribution

7.1.2.1. Reinforcement Learning Model

This model performs both exploration and exploitation together. The presentation of documents in each process is exactly similar to the split model. i.e. during exploration phase we use uniform randomization of balls, where as in exploitation phase we use deterministic permutation using the current knowledge of distribution.

We have defined a factor *alpha* which helps to pick an action from the exploration and exploitation. Alpha is a binary number whose value is always equals to 0 (exploration) or 1 (exploitation). This value of this factor provides the portion of time that we spent on the exploration and the rest of time is used for the exploitation. For the experiments on this model, I have considered various values of alpha.

7.1.2.2. No-regret Learning Model

Experiment 1: For the first set of experiments on the reinforcement model and no-

regret learning models, I used top3Biased distribution as input. The patience index considered is 0.35 for all runs. The alpha values considered are 0.15, 0.25, 0.50, 0.75, 0.85 and 0.1. The total number of customers generated are 1×10^4 (Figure: 7.4) and 1×10^5 (Figure: 7.5) respectively.

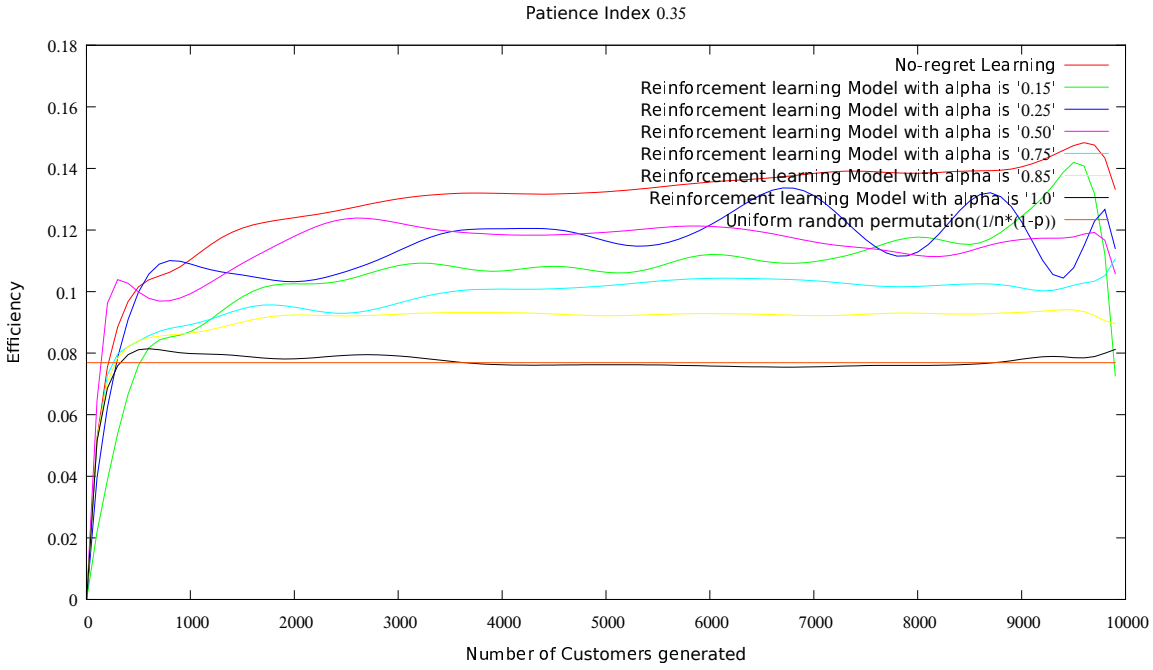

Figure 7.4.: Top3BiasedDistribution Plot for 10,000 customers

Experiment 2:

In the second set of experiments, I used twoCluster distribution as input. The patience index considered is 0.35 for all runs. The alpha values considered are 0.15, 0.25, 0.50, 0.75, 0.85 and 0.1. The total number of customers generated are 1×10^4 (Figure: 7.6) and 1×10^5 (Figure: 7.7) respectively.

Experiment 3:

For the third set of experiments on these models, I used nearUniform distribution as input. The patience index considered is 0.35 for all runs. The alpha values considered are 0.15, 0.25, 0.50, 0.75, 0.85 and 0.1. The total number of customers generated are

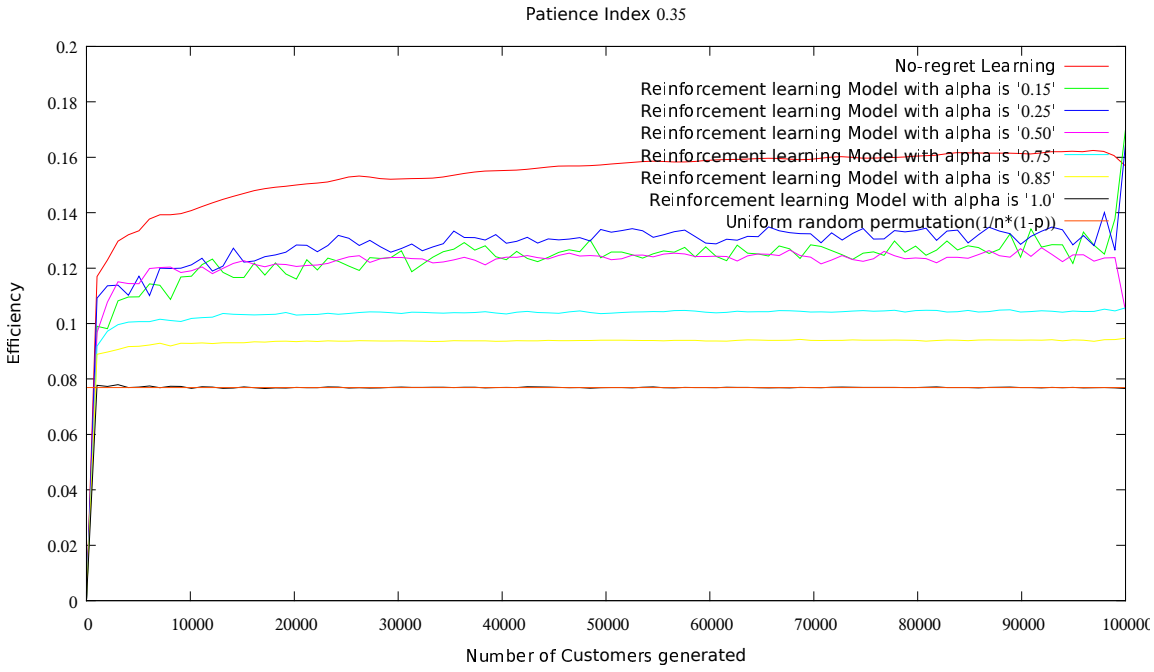

Figure 7.5.: Top3BiasedDistribution Plot for 100,000 customers

1×10^4 (Figure: 7.8) and 1×10^5 (Figure: 7.9) respectively.

Observations:

Reinforcement Learning Model: This model continuously performs the exploration and exploitation process together according to the parameter *alpha*. This means the model is spending some time on the exploration even after gaining the sufficient knowledge. This lessens the efficiency of the system. The success of this model depends on identifying the best alpha value. This is the limitation of this method.

No-regret Learning Model: This model outperforms the previous model. Unlike reinforcement learning model, this doesn't have any specific tuning parameter alpha. This approach is appears to be feasible solution. But, as this model

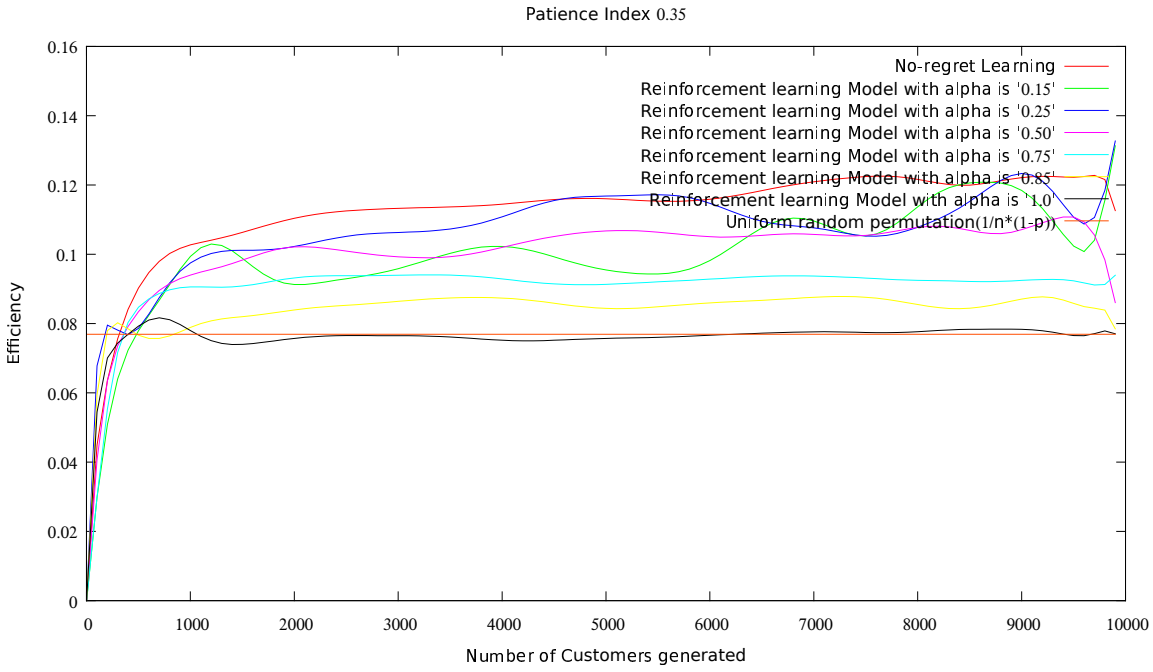

Figure 7.6.: TwoCluster Distribution plot for 10,000 customers

is totally dependent on probabilistic permutation, some times it is possible to show the bad items in lower ranking positions. This again lowers the efficiency of system. This suggests us that this model is yet to be optimized. The other drawback of this model is the slow rate of convergence. This is because the model is totally dependent on the positive events or sales. We can't leverage the negative actions of user into the feedback. If the user rejects any document this model fails to leverage it into the ranking. If we can use these actions into the ranking that could give us best convergence.

7.1.3 Learning Automata Based Ranking Model

To measure the correctness of the approach, I used performance metrics like $P@10$, $NDCG$ and Kendall tau rank correlation. Here, we are solving the ranking prob-

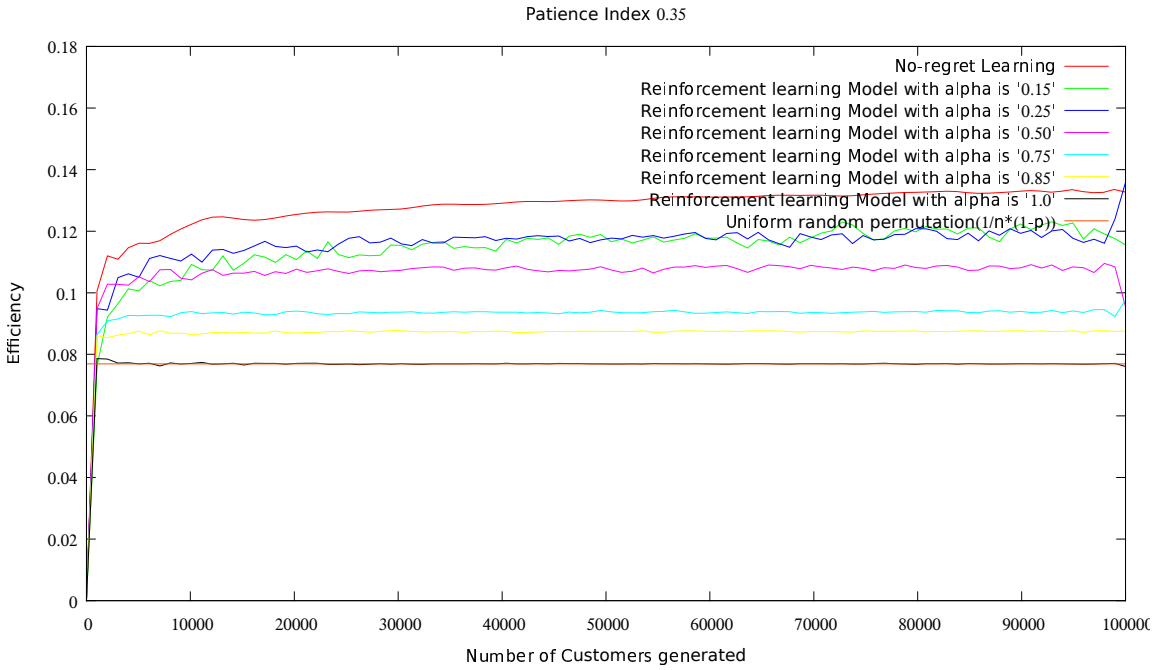

Figure 7.7.: TwoCluster Distribution plot for 100,000 customers

lem, so Kendall tau correlation has less significance. But the other two metrics are significant in identifying the correctness of results.

Experiment 1:

The first experiment we used the *top3Biased* distribution as the input. I ran the model with patience index 0.35 and I generated 5000 customers in each run. I compared the permutation to the model converged with the ideal permutation to compute the metric. Ideal permutation is simply ordering the items in non-increasing order of the input customer distribution.

The figures represents the plots of precision@10 (Figure:7.10), DCG (Figure:7.11), NDCG (Figure:7.12) and Kendall tau Value (Figure:7.13).

Experiment 2:

For this experiment we used the *twocluster distribution* as the input. We ran the model with same patience index 0.35 and generated 5000 customers in each run. I

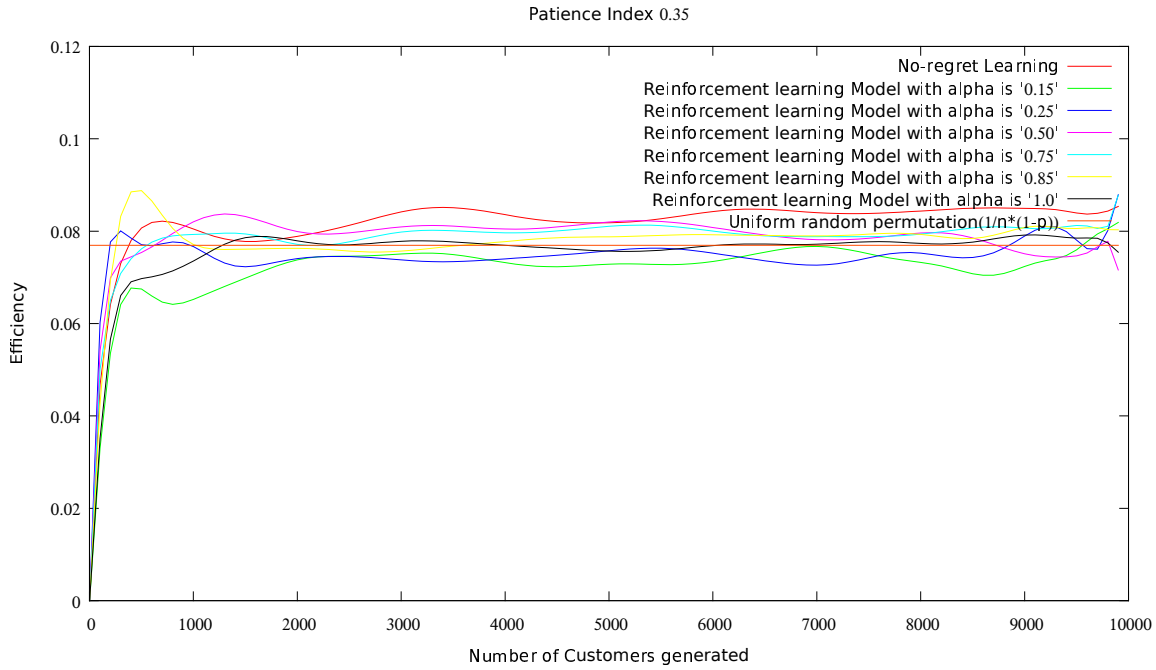

Figure 7.8.: NearUniform Distribution plot for 10,000 customers

compared the permutation the model converged to with the ideal permutation to find the correctness.

The figures represents the plots of precision@10 (Figure:7.14), DCG (Figure:7.15), NDCG (Figure:7.16) and Kendall tau Value (Figure:7.17).

Experiment 3:

For this experiment we used the *NearUniform distribution* as the input. I ran the model with the patience index 0.35 and generated 5000 customers. I have compared the permutation which the model converged with the ideal permutation to find the correctness.

The figures represents the plots of precision@10 (Figure:7.14), DCG (Figure:7.15), NDCG (Figure:7.16) and Kendall tau Value (Figure:7.17).

Observations:

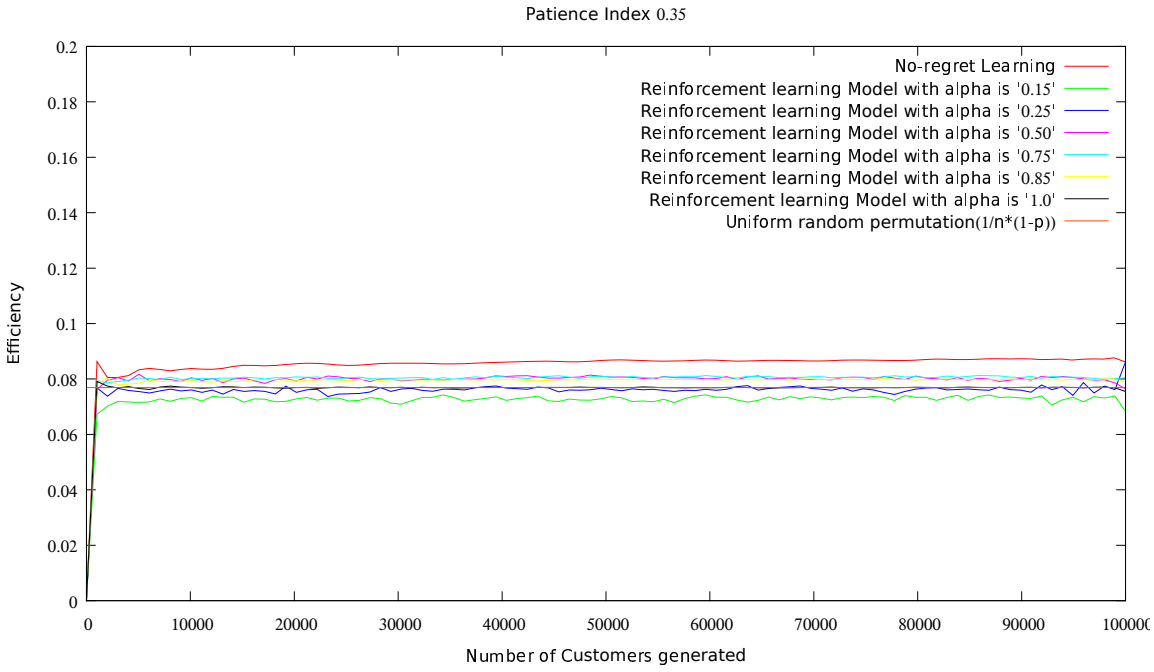

Figure 7.9.: NearUniform Distribution plot for 100,000 customers

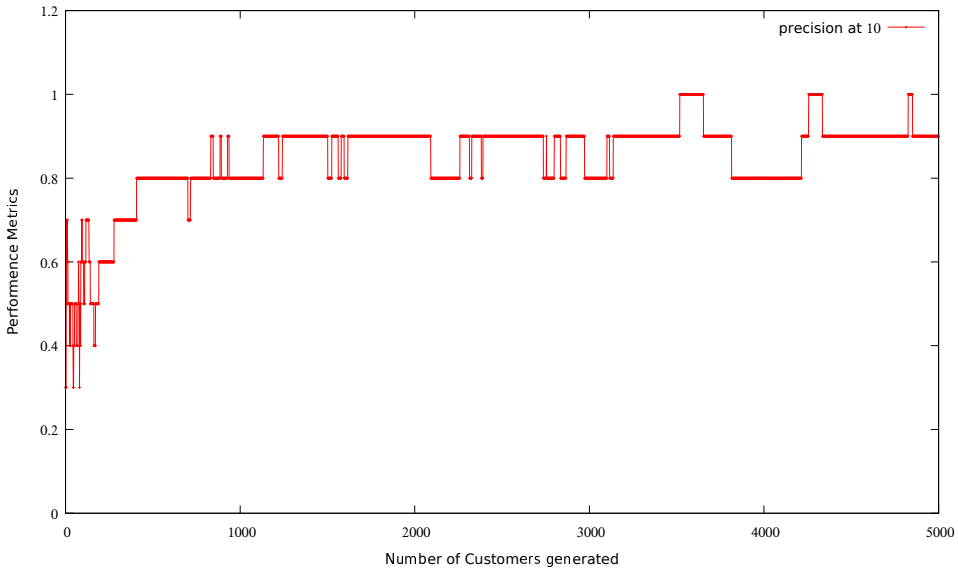

Figure 7.10.: Top3BiasedData precision@10 and patienceIndex 0.35

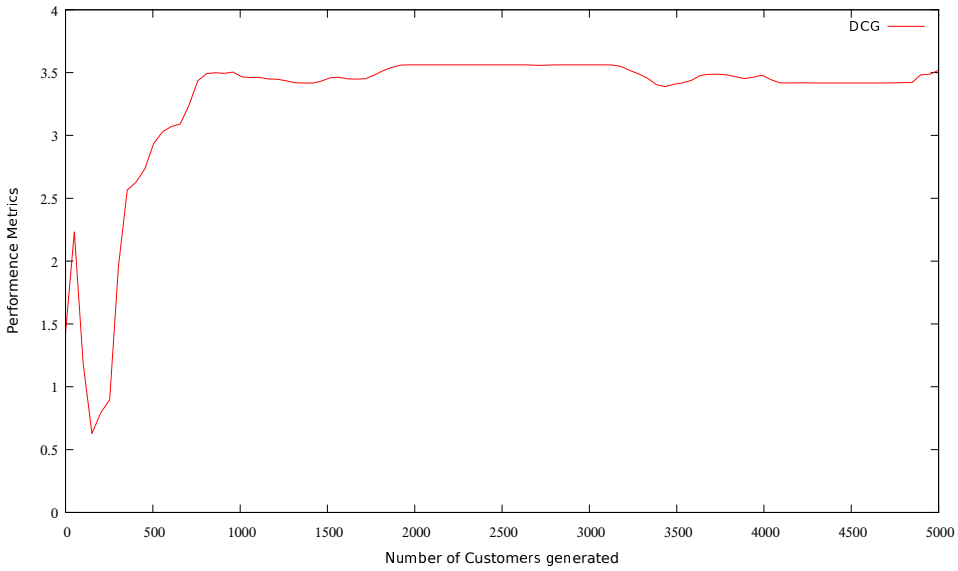

Figure 7.11.: Top3BiasedData DCG and patienceIndex 0.35

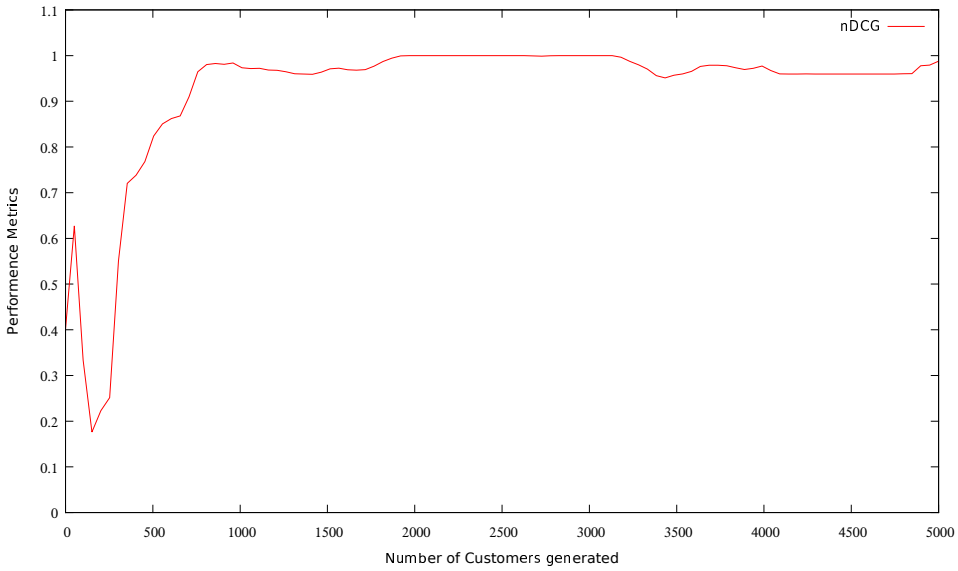

Figure 7.12.: Top3BiasedData NDCG and patienceIndex 0.35

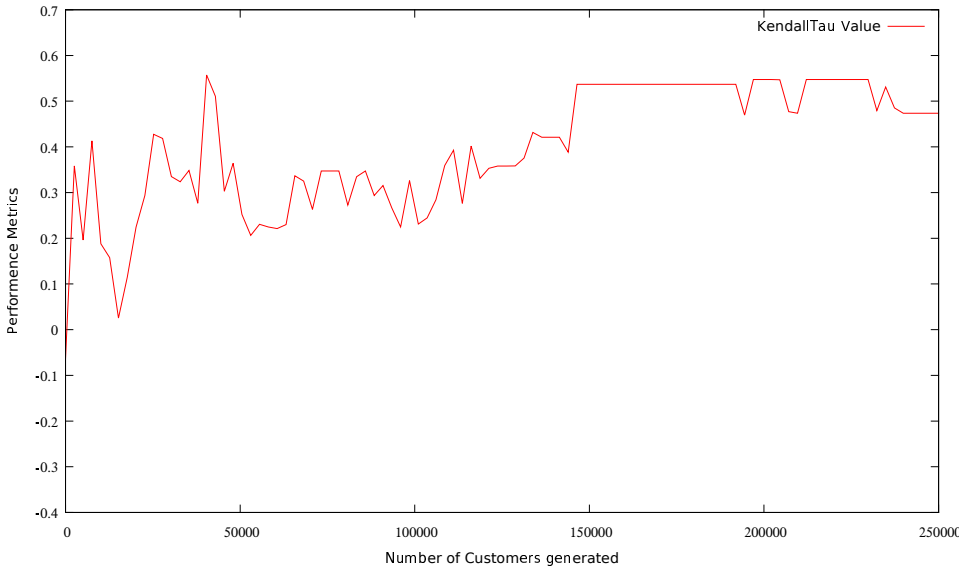

Figure 7.13.: Top3BiasedData kendallTau and patienceIndex 0.35

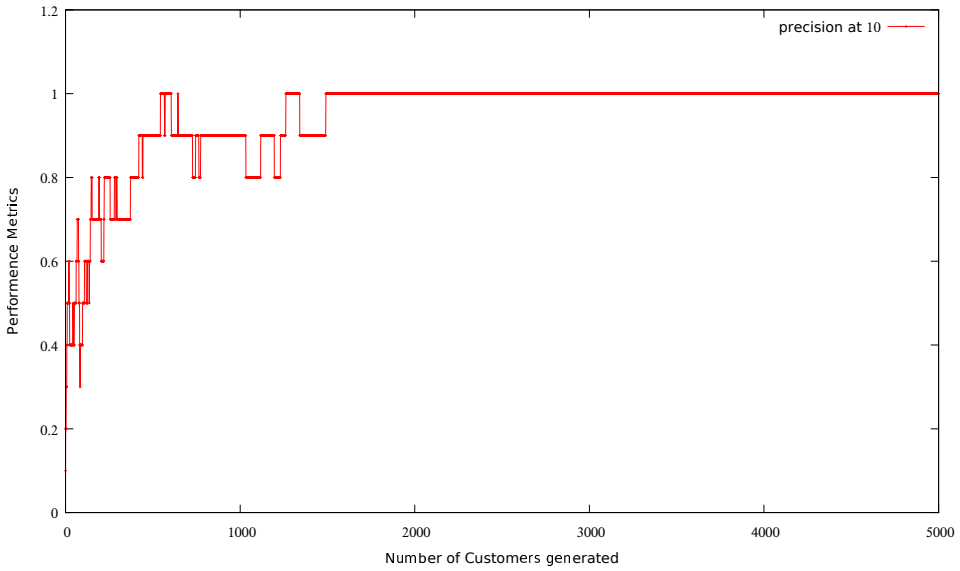

Figure 7.14.: TwoClusterDist precision@10 and patienceIndex 0.35

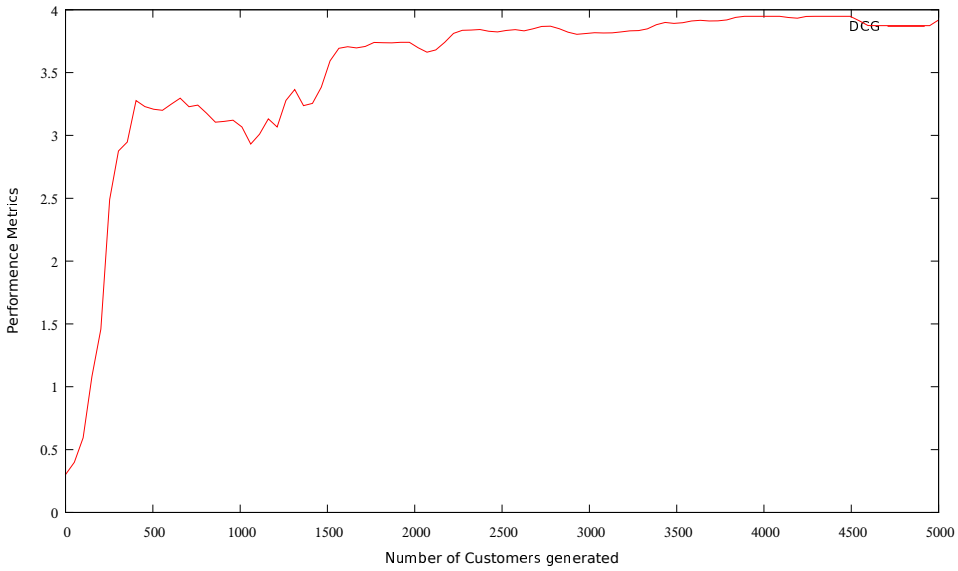

Figure 7.15.: TwoClusterDist DCG and patienceIndex 0.35

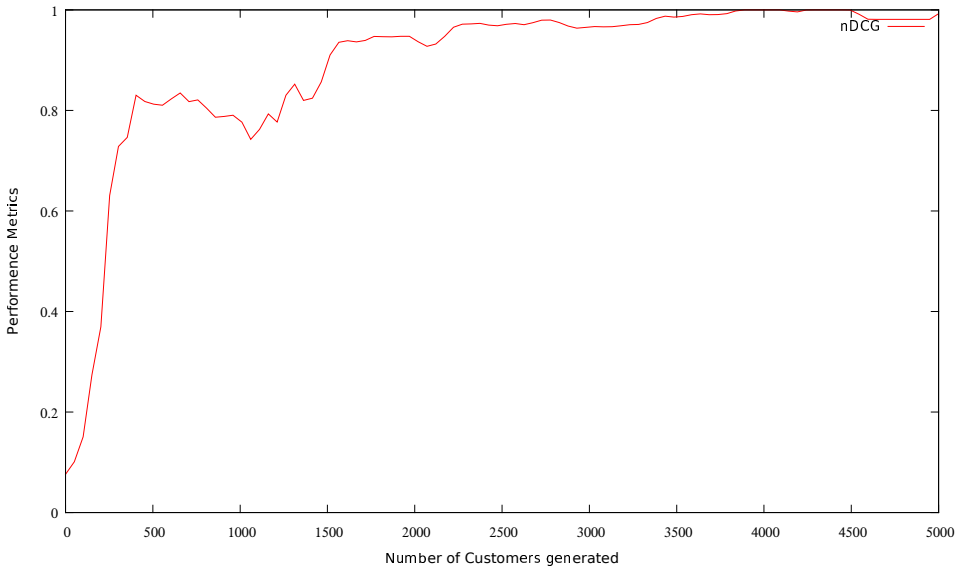

Figure 7.16.: TwoClusterDist NDCG and patienceIndex 0.35

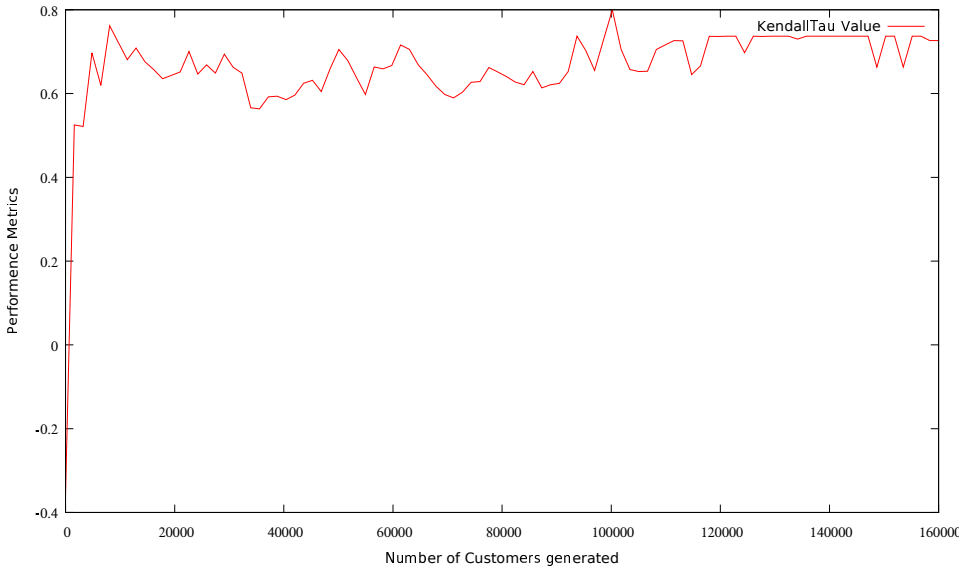

Figure 7.17.: TwoClusterDist kendallTau and patienceIndex 0.35

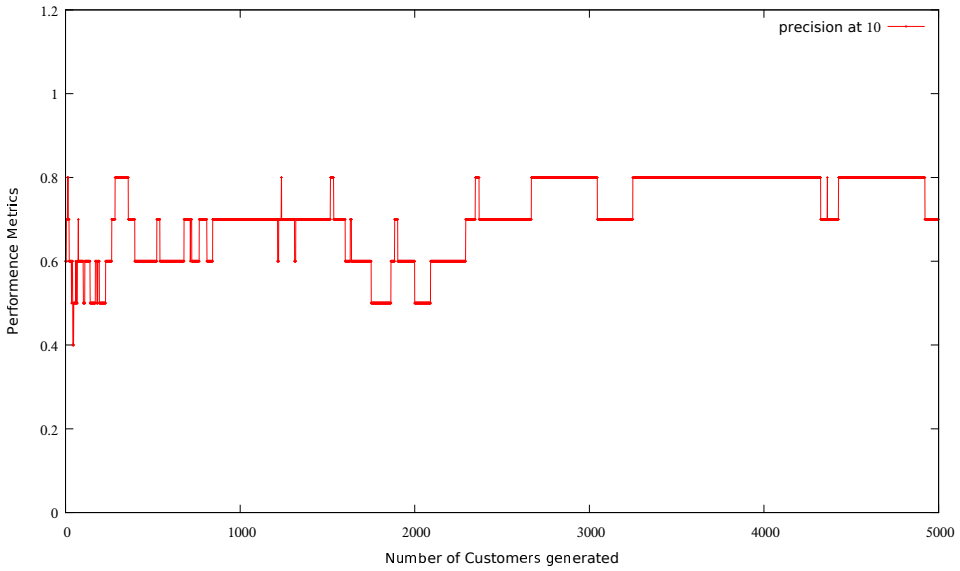

Figure 7.18.: NearUniformDistribution precision@10 and patienceIndex 0.35

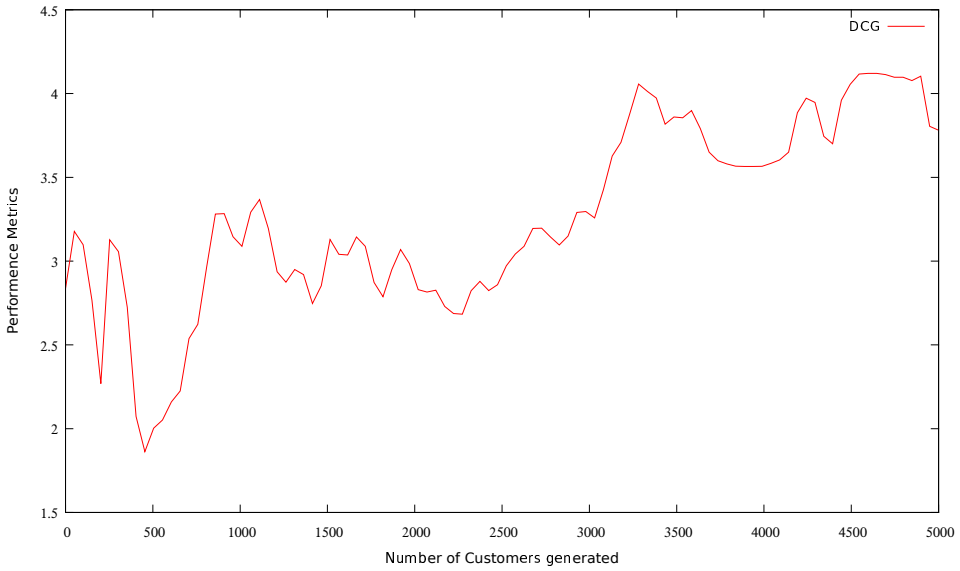

Figure 7.19.: NearUniformDistribution DCG and patienceIndex 0.35

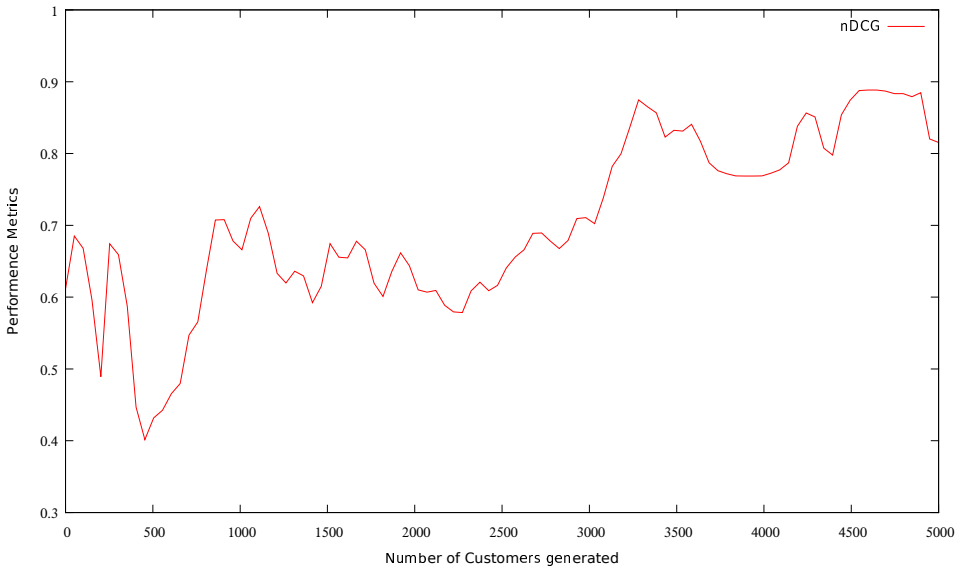

Figure 7.20.: NearUniformDistribution NDCG and patienceIndex 0.35

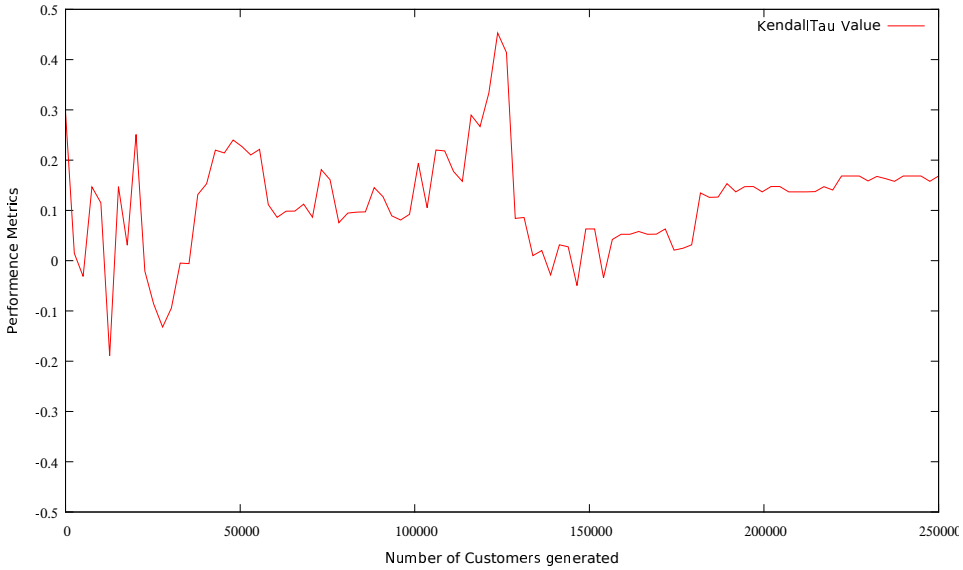

Figure 7.21.: NearUniformDistribution kendallTau and patienceIndex 0.35

This model has the best convergence. In this model the $pVector$ will be gets updated in all possible scenarios. The $pVector$ moves towards the action selected if the response is positive and moves in other way in case of penalty. Hence, this model convergences quickly compare to previously discussed models. In all of the experiments I have generated total 5000 customers. From the above figures we can observe that model learns distribution in the very beginning with in a very few iterations (first 1000 customers).

8 FUTURE WORKS

The proposed Learning Automata Based Ranking Model in this thesis is flexible to implement in all types of environments, which means this model can be extended to non-binary environments like Q-model (where the environment responses are finite set of values) and S-model (where the environment responses belongs to an interval). This flexibility makes it suitable to extend further to more complicated real world web search usage. Especially, we can improve the model convergence rate by extending it to different types of environments. Currently, i have rewarded the result having user click and uniformly penalized the ignored results. But, here we can improve the performance of this method by introducing the magnitude to the reward/penalty. We can consider the amount of time user spent on each document as the factor to decide the reward magnitude. We can also add the distinct penalty values to each document based on its position in the ranking order.

The proposed model can be made more realistic by extending it to a multiple click scenario of web search. In all my experiments I have assumed a single click search endeavor, this model suits well for scenarios having more than one click in each search process.

In this model, the update process of action probability vecotor (pVector) is based on the Bush-Mosteller scheme. This scheme is an expedient scheme. The expedient schemes produces the Markov process that are ergodic in nature. Hence, this process don't have any absorbing state and stabilizes the pVector to an equilibrium point which helps to build the ranking. As this update process doesn't create absorbing barriers, this has the ability to adopt change. This makes the proposed method suitable for the Non-stationary environments and periodic environments too.

LIST OF REFERENCES

LIST OF REFERENCES

[1] Ellen M. Voorhees and Donna K. Harman. *TREC: Experiment and Evaluation in Information Retrieval (Digital Libraries and Electronic Publishing)*. The MIT Press, 2005.

[2] Yisong Yue, Rajan Patel, and Hein Roehrig. Beyond position bias: Examining result attractiveness as a source of presentation bias in clickthrough data. In *Proceedings of the 19th International Conference on World Wide Web*, WWW '10, pages 1011–1018, New York, NY, USA, 2010. ACM.

[3] Andrew Turpin and Falk Scholer. User performance versus precision measures for simple search tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 11–18, New York, NY, USA, 2006. ACM.

[4] Eugene Agichtein, Eric Brill, Susan Dumais, and Robert Ragno. Learning user interaction models for predicting web search result preferences. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 3–10, New York, NY, USA, 2006. ACM.

[5] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 133–142, New York, NY, USA, 2002. ACM.

[6] Charles Kemp and Kotagiri Ramamohanarao. Long-term learning for web search engines. In *Proceedings of the 6th European Conference on Principles of Data Mining and Knowledge Discovery*, PKDD '02, pages 263–274, London, UK, 2002. Springer-Verlag.

[7] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison Wesley, 2011.

[8] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[9] M. E. Maron and J. L. Kuhns. On relevance, probabilistic indexing and information retrieval. *J. ACM*, 7(3):216–244, July 1960.

[10] S. E. Robertson. Overview of the okapi projects [introduction to special issue of journal of documentation. *Journal of Documentation*, 53:37, January 1997.

[11] Tao Tao and ChengXiang Zhai. Regularized estimation of mixture models for robust pseudo-relevance feedback. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 162–169, New York, NY, USA, 2006. ACM.

[12] Cheng Xiang Zhai, William W. Cohen, and John Lafferty. Beyond independent relevance: Methods and evaluation metrics for subtopic retrieval. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 10–17, New York, NY, USA, 2003. ACM.

[13] Chengxiang Zhai and John Lafferty. Model-based feedback in the language modeling approach to information retrieval. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, pages 403–410, New York, NY, USA, 2001. ACM.

[14] Azadeh Shakery and ChengXiang Zhai. A probabilistic relevance propagation model for hypertext retrieval. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, CIKM '06, pages 550–558, New York, NY, USA, 2006. ACM.

[15] Filip Radlinski, Madhu Kurup, and Thorsten Joachims. How does clickthrough data reflect retrieval quality? In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, CIKM '08, pages 43–52, New York, NY, USA, 2008. ACM.

[16] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 232–241, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[17] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22Nd International Conference on Machine Learning*, ICML '05, pages 89–96, New York, NY, USA, 2005. ACM.

[18] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking svm to document retrieval. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 186–193, New York, NY, USA, 2006. ACM.

[19] Brian T. Bartell, Garrison W. Cottrell, and Richard K. Belew. Automatic combination of multiple ranked retrieval systems. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 173–181, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[20] William W. Cohen, Robert E. Schapire, and Yoram Singer. Learning to order things. *J. Artif. Int. Res.*, 10(1):243–270, 1999.

[21] D. Mladenic. Personal webwatcher: Implementation and design. *Technical Report IJS-DP-7472, Department for Intelligent Systems, J.Stefan Institute*, 1996.

[22] Henry Lieberman. Letizia: An agent that assists web browsing. In *INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE*, pages 924–929, 1995.

[23] Jeremy Goecks and Jude Shavlik. Learning users' interests by unobtrusively observing their normal behavior. In *Proceedings of the 5th International Conference on Intelligent User Interfaces*, IUI '00, pages 129–132, New York, NY, USA, 2000. ACM.

[24] Masahiro Morita and Yoichi Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 272–281, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[25] Joseph A. Konstan, Bradley N. Miller, David Maltz, Jonathan L. Herlocker, Lee R. Gordon, and John Riedl. Grouplens: Applying collaborative filtering to usenet news. *Commun. ACM*, 40(3):77–87, 1997.

[26] Douglas W. Oard and Jinmook Kim. Implicit feedback for recommender system. Technical report, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer, 1998.

[27] Laura A. Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in www search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 478–479, New York, NY, USA, 2004. ACM.

[28] Andrei Broder. A taxonomy of web search. *SIGIR Forum*, 36(2):3–10, September 2002.

[29] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, WSDM '08, pages 87–94, New York, NY, USA, 2008. ACM.

[30] Georges E. Dupret and Benjamin Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 331–338, New York, NY, USA, 2008. ACM.

[31] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: Estimating the click-through rate for new ads. In *Proceedings of the 16th International World Wide Web Conference(WWW-2007)*, January 2007.

[32] Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 1–10, New York, NY, USA, 2009. ACM.

[33] Filip Radlinski and Thorsten Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pages 1406–1412. AAAI Press, 2006.

[34] Ricardo A Baeza-Yates and Berthier Ribeiro-Neto. Modern information retrieval. 1999.

[35] KALERVO JARVELIN and JAANA KEKALAINEN. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, October 2002.

[36] Emine Yilmaz, Javed A. Aslam, and Stephen Robertson. A new rank correlation coefficient for information retrieval. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 587–594, New York, NY, USA, 2008. ACM.

[37] Kumpati S. Narendra and Mandayam A. L. Thathachar. *Learning Automata: An Introduction*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[38] Kaddour Najim and Alexander S Poznyak. *Learning automata: theory and applications*. Elsevier, 2014.