The Pennsylvania State University

The Graduate School

The Harold and Inge Marcus
Department of Industrial & Manufacturing Engineering

An Analysis of Factors Predicting Click Through Rate During Web Searching Using Neural Networks

A Thesis in

Industrial Engineering and Operation Research

by

Ying Zhang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2008

The thesis of Ying Zhang was reviewed and approved* by the following:

Bernard J. Jansen
Assistant Professor, College of Information Sciences and Technology
Thesis Co-Advisor

Enrique del Castillo
Distinguished Professor of Industrial Engineering
Thesis Co-Advisor

Richard J. Koubek
Professor of Industrial Engineering
Head of the Harold and Inge Marcus Department
of Industrial and Manufacturing Engineering

***Signatures are on file in the Graduate School.**

ABSTRACT

In this research, I investigate the interactions between users and search engines during Web searching using Neural Networks (NNs) and data from two Web search engine transactional logs. The goal of this research is to identify factors that significantly affect the click through rate (CTR) of Web searchers. CTR, a metric for measuring customer satisfaction, is calculated by dividing the number of links that a searcher clicks by the number of the links that are delivered. The purpose of this research is to leverage user-system interaction data to improve CTRs and to design more efficient ranking and searching algorithms. The analysis in this research consists of two parts: basic data analysis and extended data analysis. Basic data analysis shows that users spend less effort on retrieving their information needs later in the day and on the weekend. Most people prefer using the Internet Explorer browser, selecting informational links, and searching the Web vertically. In the extended data analysis, two NNs are utilized: Multilayer Backpropagation Neural Network (MLPN) and Radial Basis Function Neural Network (RBFN). We investigate the characteristics and quality of these two kinds of networks by screening out the nonlinear behavior in the search logs. We then use the MLPN, by virtue of its higher efficiency compared with the RBFN, to detect the influence of significant information on the CTR. Our findings show that the number of organic links viewed, the type of vertical, and the type of user intent have a negative correlation with the CTR. However, we find that the number of queries reformulated by users, searching time, average query length, type of browser, and the rank of links sum is shown in the Searching Result Page (SERP). All are positively correlated with the CTR. Based on these results, some recommendations for improving the performance of the searching algorithms are provided. Implications of current research will present a new direction for future research.

Table of Contents

List of Figures

List of Tables

Acknowledgement

This has been a period of great challenge and enrichment, but at the same time has been a wonderful experience, and one for which I am grateful.

I am deeply grateful to my thesis adviser, Dr. Jim Jansen, for his kindly patience and invaluable guidance through out this period. With his unceasing encouragement, I have learned to build the confidence in myself to go beyond my boundaries. I also greatly appreciate Dr. Castillo for his patience and inspired advice, making my thesis an improved body of work.

I am also grateful to Mr. Riedel. His guidance and support is tremendous. His generous help made the beginning of my career in industrial engineering much easier and made my life in the U.S. much more enjoyable. I truly appreciate him and his family.

My friends at Penn State, particularly in IE department, Jie Zhang, Zhi Yang, Jing Yin, J. R. Humphrey, Terek Maalouf, Ana Karina Miranda, Zhibiao Lai and Dawei Xu, and Dr. Jeya Chandra greatly supported me and provided huge help during this period. To all these and to other friends, I am grateful.

As it is with almost everything, I owe so much to my parents. Without their unconditional faith, love, and support, I could have never reached this step.

Thank you!

Chapter 1

Introduction and Overview

1.1 Introduction

Search engines are information retrieval systems designed to help users find information stored on the World Wide Web, inside a corporate or proprietary network, or in a personal computer. According to Hawking (2006), large search engines operate geographically distributed data centers. Within a data center, services are built from clusters of commodity personal computers. These clusters or individual servers can be dedicated to specialized functions such as crawling, indexing, query processing, snippet generation, link-graph computations, result caching, and insertion of advertising content. Search engines allow users to ask for content that meets specific criteria (typically those containing a given word or phrase) and retrieve a list of items that match those criteria. This list is often sorted to a certain degree by relevance of the results. Search engines use regularly updated indexes to operate quickly and efficiently.

Without further qualification, the term “search engine” usually refers to a Web search engine. Different selection and relevance criteria may apply in different environments or for different uses. Other kinds of search engines are enterprise search engines, personal search engines, and mobile search engines. In addition, some search engines mine data available in newsgroups, databases, or open directories.

The usefulness of a search engine depends on the relevance of the result set that it gives back. While there may be millions of Web pages that include a particular word or a phrase, some pages may be more relevant, popular, or authoritative than the others. Most search engines employ methods to rank the results in order to provide the "best" or most relevant results first. How a search engine decides which pages are the best matches and in what order to show the results varies widely from one engine to another. The methods also change over time as Internet use changes and new techniques evolve. Therefore, the evaluation of searching efficiency is a critical and ongoing research area.

One important resource for evaluating the performance of a search engine is the transactional log, an electronic record of interactions that occur during a searching episode between a Web search engine and a user (Jansen, 2006). Generally, a transactional log contains the data concerning the client computer's Internet Protocol (IP) address, log on time, user query, and query length, as well as some searching behaviors of online users. Using this information, we can begin to unravel the behaviors of online users, such as identifying query patterns and logging on density. Importantly, transactional log data has information with substantial potential for commercial value.

For example, the transactional log data contains information on the click through rate (CTR), which is a metric for measuring customer satisfaction. CTR is an important element reflecting the quality and effectiveness of commercial search engines and online advertising, such as those of sponsored search campaigns. After simple data arrangement and calculation, the information stored in transactional logs could reflect the CTR. We believe that information stored in transactional logs can be used not only to identify basic descriptive factors but also to develop advanced inference models (i.e., based on certain information, it is able to predict the CTR).

In this research, we aim to discover the relationship between some basic information (browser type, query length, etc.) and CTR, as well as to identify potentially significant information that can predict CTR.

1.2 Search Engine Topology

In this section, we present how search engines work and how users submit queries to search the Internet. Generally, a search engine surveys or “crawls” the Web, searching and indexing information from a large number of Web pages. Web search engines work by storing information in the form of a large number of Web pages, which they retrieve from the Web itself. These pages are retrieved by a Web crawler (also known as a spider) — an automated Web browser that follows every link it discovers. The crawler initializes the queue with one or more seed links. Crawling proceeds by making an HTTP request to fetch the page of the first link in the queue.

When the crawler fetches the pages, it scans the contents for links to other Web pages and adds each previously unseen link to the queue. Exclusions can be made by the use of robots.txt (Hawking, 2006). The contents of each page are then analyzed to determine how it should be indexed; for example, words are extracted from the titles, headings, or special fields called Meta tags. Data about Web pages are stored in an index database for use in later queries.

When a user enters a query into a search engine, the engine examines its index and provides a listing of the best-matching Web pages according to its criteria, along with a short summary containing the document's title and sometimes parts of the text. Figure 1-1 illustrates the typical search engine’s retrieval process.

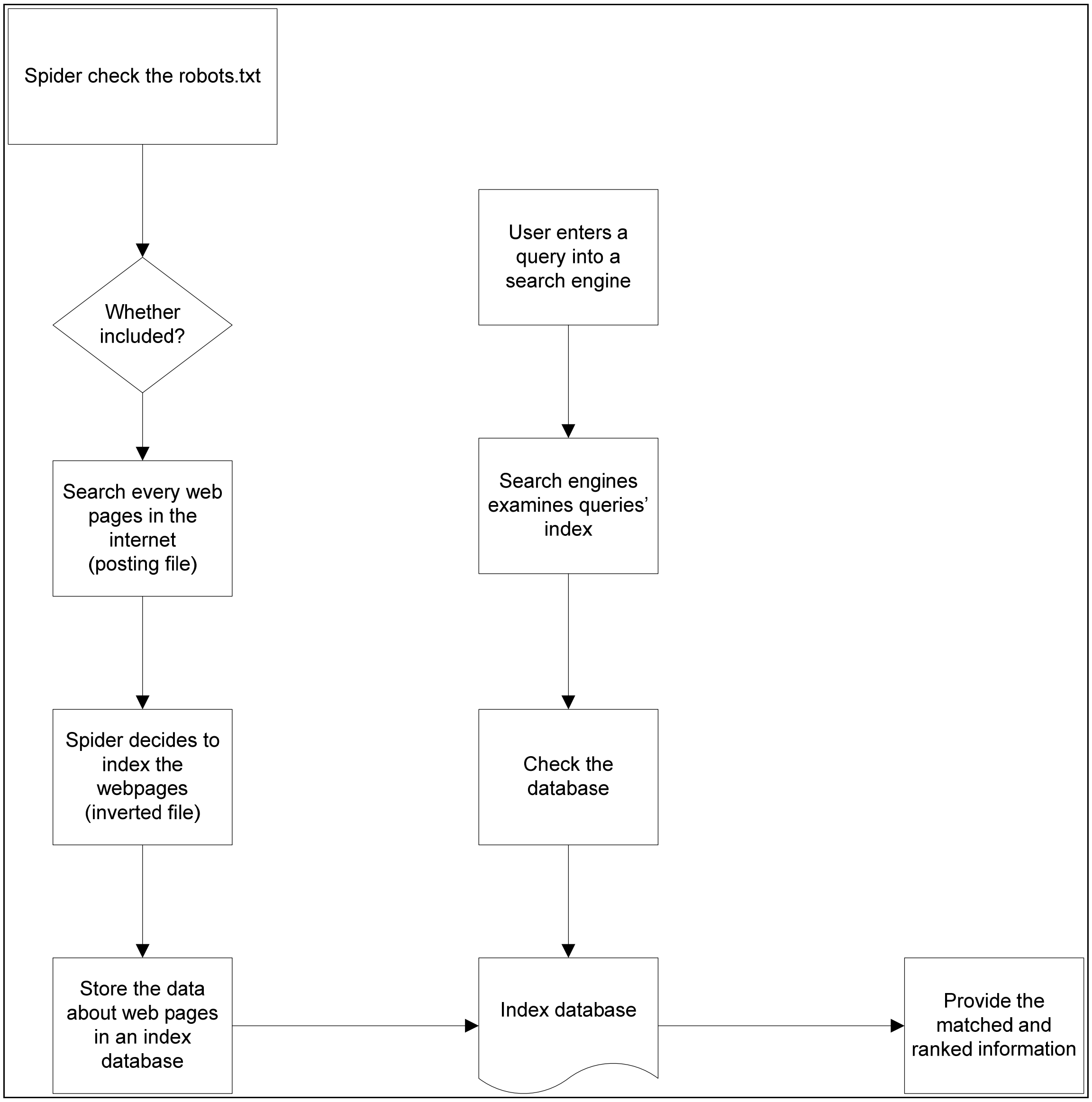

Spider check the robots.txt
Whether included?
Search every web pages in the internet (posting file)
Spider decides to index the webpages (inverted file)
Store the data about web pages in an index database
User enters a query into a search engine
Search engines examines queries' index
Check the database
Index database
Provide the matched and ranked information

Figure 1-1: Retrieval process of typical search engine

1.3 Motivation

As comprehensive records of searchers' online behaviors, transactional logs provide important data for studying Web information retrieval and searching; however, such logs produce enormous amounts of data that companies and researchers must analyze in order to decipher the complex behaviors of searchers. For example, Nielsen/NetRatings measures the search behavior of approximately 500,000 people worldwide (Sullivan, 2006). The amount of user interaction data in the form of a search engine transactional log is an order of magnitude larger than anything available in non-Web-search settings, and analyzing transactional logs of this size poses significant challenges. By using the aggregated behavior of large numbers of users, however, we can grasp trends of behaviors and erase abnormal individual behaviors. This ability is especially beneficial when individual users behave irrationally or maliciously.

To date, much of the research using transaction log analysis has followed three, largely descriptive lines. First, some researchers have investigated transactional log analysis as an approach for investigating both server and client-side interactions (Park, Bae and Lee, 2005). Other researchers are interested in the semantic interpretation of user text files (Heckerman and Horvitz, 1998), and still others (Beitzel, Jensen and Chowdhury, 2004) are focused on the temporal aspects of the data. In order to take search log analysis to a more meaningful level, we must develop models that use automated data analysis techniques to uncover system-user interactions.

In the field of information retrieval, data mining has become more popular (Meghabghab & Kandel, 2004; Almpanidis, Kotropoulos & Pitas, 2007). Some experts have focused on historical data to do the analysis, using statistical methods to analyze user behavior in order to find valuable information in transactional logs (Spink, Jansen, Wolfram, & Saracevic, 2002; Marable, 2003; Jansen & Spink, 2006). These methods focus on information clustering, classification, or basic statistical analysis aspects using complicated data mining techniques. None of these data-mining techniques, however, provide an efficient model for identifying user-system interactions based on evidence of

CTR available from Web search engine transactional logs. Because search engine transactional logs are technically discrete time series data sets, they pose a substantial challenge to creating an efficient model for analyzing Web searches. In addition, data sets recording searchers' behaviors are random and involve many subjective elements. Considering these challenges, neural networks (NNs) may be useful for system identification because these networks are capable of learning complex mappings from a set of examples.

Due to the function approximation properties of NNs as well as their inherent adaptation features, NNs present an appealing alternative for the modeling of nonlinear systems (Giles, Lawrence & Tsoi, 2001). Unfortunately, few studies of NN application to the Web-related information, especially to Web search engines, currently exist. Therefore, the objective of this thesis is to apply NNs to daily Web search transactional logs in order to analyze user online searching behaviors. Commercial search engine companies could utilize user-system interactive data to improve the CTR and design more efficient searching algorithms.

1.4 Thesis Overview

This thesis is structured as follows. Chapter 2 summarizes concepts and previous work related to the use of Web transactional logs to investigate user behaviors. In Chapter 3, we introduce the basic theories and training algorithms underlying two major feedforward NN methods – MLPN (Multilayer Backpropagation Neural Network) and RBFN (Radial Basis Function Neural Network). In Chapter 4, we use basic data analysis methods to explore the basic characteristics of the information represented in the transactional log. We then collect the necessary data sets(training, testing, and evaluating) to build the corresponding NN, explore the constructed NN using prepared data sets, and analyze their characteristics by changing parameters. After that, we present a sensitivity analysis of the input neurons on the CTR based on the most efficient network. In Chapter 5, we highlight the importance of the models utilized before concluding with discussion of the findings.

Chapter 2

Literature Review

Search engines continue to attract a large number of Web searchers and consistently rank among the most heavily visited Web sites (Nielsen/Netrating, 2002). On the consumer side, surveys indicate that search engines are the most important promotional method used by e-commerce sites, and furthermore, they represent the most common way that new e-commercial sites market themselves. Most Web search engines are commercial ventures supported by advertising revenue, and as a result, some employ the controversial practice of allowing advertisers to pay money to have their listings ranked higher in search results. The search engines that do not accept money for their search engine results make money by running search related advertisements alongside the regular search engine results. In some sense, the search engines make money every time someone clicks on one of these advertisements. The study of search engines is important both in terms of improving search capabilities for individual users as well as promoting companies' profits.

2.1 Information Retrieval

The majority of search engines work as a form of information retrieval (IR), and given their popularity and heavy use, efficiency in retrieving information is essential. IR on the Web is an interactive and iterative process that includes presenting, storing, searching, and finding information that is relevant to a user's stated criteria (Ingwersen, 1996).

As Swanson (1977) points out, trial and error has an important role in the IR process. An initial request is a guess about the attribute of desired documents, after which the response of the IR system prompts the user to revise the initial guess for another try. For search engines, users enter the keywords they know in their initial queries. If the initial query does not return the expected search results, then users submit their second best keywords. Therefore, despite the perception that Web searching is simple and easy, approximately half of all Web users find they must reformulate their initial queries. For example, 52% of the users in the 1997 Excite data set and 45% of the users in the 2001 Excite data set made modifications to their initial queries (Spink, Jansen, Wolfram and Saracevic, 2002). Bruza and Denis (1997) categorized query reformulation into 11 types and found that users frequently repeat queries they had already submitted. Additionally, they identified the following main categories of user query reformulations, in descending order of frequency: term substitutions, additions, and deletions. From this point of view, the searching process is the iterative interaction between users and system negotiating perception and criteria of desired information.

Rieh (2006) expressed that the interactive information retrieval process was cyclical such that both query terms and searching results are constantly selected, evaluated, and modified. This understanding of query reformulation sequences poses a significant research problem. Based on information retrieval interaction models, researchers have presumed that query reformulation results from the interaction between users and IR systems. The analysis of query modification sequences identified eight distinct patterns: specified, generalized, parallel, building block, dynamic, multitasking, recurrent, and format reformulation.

The framework of interactive IR models allows for an investigation of query reformulations in terms of interacting with the system, interpreting search results, and shifting strategies beyond comparison of search queries requested to search results received. Therefore, the IR model has become another important branch of study in the search engine literature.

2.2 Web Search Engine Evaluation

Besides the work on IR, another major theme of the study is information relevance evaluation. The interface design of Web search engines appears to be fairly standardized: a search box for 20-30 characters and a search button next to it. This standard interface, however, leads to numerous questions. How can this simply designed interface satisfy users' searching demands? How should user response be investigated, and importantly, what aspects of user response, exactly, should be investigated? Researchers have employed a variety of methods and approaches to address these questions.Many researchers have explored accuracy or precision of various Web search engines, achieving varying results. For example, Chu and Rosenthal (1996) used three Web search engines (AltaVista, Excite and Lycos) and 10 queries to examine precision via a three-level relevance score (relevant, somewhat relevant and irrelevant) for the top 10 links. The authors reported that Alta Vista outperformed the other search engines in search facilities and retrieval performance. In the same year, Ding and Marchionini (1996) evaluated three search engines using five topics. Along with including links to other relevant documents, the researchers employed a six-point scale to calculate three different types of precision. They found no significant differences among the search engines.

Nicholson (2000) replicated this study, employing chaos theory to highlight possible patterns.

Other studies expanded to include more search engines, more search requests, and varying measurement tools. For instance, Gordon and Pathak (1999) evaluated eight search engines (AltaVista, Excite, Infoseek, Open Text, HotBot, Lycos, Magellan, and Yahoo!) using 33 queries and the top 200 links. The researchers used a four-level relevance judgment (highly relevant, somewhat relevant, somewhat irrelevant, and highly irrelevant) and one evaluator (i.e., the person that had requested the information search). Using an analysis of variance, AltaVista, Open Text, and Lycos had statistically higher precision and recall compared with the other five search engines. In an interesting twist for search engine evaluation, the researchers had expert searchers translate the information needed into a series of queries, selecting the optimal query over a set period. This approach of using expert searchers is rarely employed when evaluating end user searching systems, such as Web search engines.

Other researchers have focused their evaluations on the links generated by search engines. Jansen (2000), for instance, compared links among several search engines, noting that regardless of query formulation, there was approximately 60% overlap on search results. Similarly, Eastman (2002) reported that links of searches using a variety of query formulations with several Internet search engines showed that strategies intended to give narrower and more precise results might not actually give improvements in precision. Eastman and Jansen (2003) reported that query formulation had no significant effect on coverage, ranking, or relevance of links on Google, MSN, or AOL search engines. More recently, Jansen and Molina (2006) examined the effectiveness of five different types of search engines in response to e-commerce queries by comparing the engines' quality of e-commerce links using topical relevancy ratings. The findings indicated that any links retrieved using an e-commerce search engine were significantly better than those obtained from most other engines types, but did not significantly differ from links obtained via a Web directory service.

Methods of assessing search engines' effectiveness have also garnered attention. Can, Nuray, and Sevdik (2004) proposed an automatic Web search engine evaluation method as an efficient and effective assessment tool. Using eight Web search engines (AllTheWeb, AltaVista, Hotbot, InfoSeek, Lycos, MSN, Netscape, and Yahoo!), 25 queries, and the top 20 links, the researchers used binary user relevance judgments determined by the information requester. Using Pearson's r correlation, the researchers reported that their method provided results consistent with human-based evaluations.

Although the above studies generated significant results, they are limited by their size and focus on search results. Consequently, they cannot provide a comprehensive evaluation of user behaviors.

2. 3 Sponsored Links vs. Organic Links

Search engine results pages (SERPs) contain at least two categories of links, sponsored and non-sponsored or organic links. Sponsored links are listed because a company, organization, or individual purchased the key words that the searcher used in the search query, and organic links are retrieved using a proprietary matching algorithm (Jansen and Resnick, 2006). According to McCarthy (2005), paid searches accounted for 99%, 84% and 12% of the revenue for Google, Yahoo!, and AOL, respectively.

Some prior research has established a potential disconnection between the perception of sponsored listings by business and users. Users appear to be suspicious of sponsored links and may regard them as less relevant than organic links and thus may be less likely to select them. Marable (2003) reported that searchers trusted search engines to present only unbiased results on the first page, not realizing that 41% of their selections were sponsored search listings. Importantly, when the searchers were informed about the nature of the sponsored listings, participants reported negative emotional reactions.

Jansen (2006) reported that searchers had a bias against sponsored links, even when controlled for content; but when viewing the content of sponsored links, searchers ranked them just as relevant as the pages of the organic links. IProspect Inc. (2004) surveyed 1,649 Web users. Of the respondents, 60% of Google users reported organic results to be more relevant than sponsored results. Frequent users of the Web found organic results to be more relevant than sponsored results (65% to 56%). More women (43%) than men (34%) found sponsored results to be generally relevant.

2. 4 Studies of Search Engine Transactional Logs

Transactional logs are collected from a Web search engine through the selection of search sessions in which users submit several unique queries in each session. Transactional logs, as a comprehensive documentation of searchers' on line behaviors, have become an important area for information retrieval. The comprehensive nature of such logs, however, means that companies interested in user behavior on the Web are faced with enormous amounts of data to analyze.

The usefulness of transactional logs has substantial support from Zipfian's theory. With Internet or Web-based phenomena, Zipfian observed a large number of rare events (LNER) with lengthy tails based on a visual inspection of a trend line of logarithmically transformed data (Baayen, 2001). However, for those highly skewed, larger data sets representing millions of potential observations, a simple Zipf distribution may not be adequate to model the observed behavior of the electronic environmental characteristics. As sample data set increases in size, it may become increasingly skewed, which may influence the type of model used for fitting.

A more rigorous method of model fitting can determine if observed distributions differ significantly from the theoretical models. This is usually achieved through the application of goodness-of-fit testing. Two primary methods in use are the chi-square (χ^2) test and the Kolmogorov-Smirnov (K-S) test. The coefficient of variation (R^2), which is generally used for regression analysis or for comparative purposes, can also provide an indication of closeness of fit. Goodness-of-fit testing, when used for comparative purposes, can demonstrate whether one model performs better than another does. Applying goodness-of-fit testing to the hypothesis that the observed data set is significantly different from the developed model can be challenging. As one approach to this dilemma, Heckerman and Horvitz (1998) described a Bayesian approach to model the relationship between words in a user's query for assistance and the information goals of the user.

Besides the efforts of trying to fit the transactional log distribution, some researchers investigated transactional log analysis as an approach by analyzing both server and client-side interactions on the final analysis (Jansen, 2006; Jansen and Molina, 2006; Park, Bae, and Lee, 2005). For example, Park, Bae, and Lee (2005) examined characteristics and interactions with a Korean-based search engine from the perspectives of session length, query length, query complexity, and content viewed among the Web search engines.

Beitzel, Jensen, and Chowdhury (2004) reviewed a log of hundreds of millions of queries that constituted the total query traffic of a general purpose commercial Web search service. They found that query traffic from particular topical categories differed both from the query stream as a whole and from other categories. This analysis provided valuable insight for improving retrieval effectiveness and efficiency. It is also relevant to the development of enhanced query disambiguation, routing, and caching algorithms.

Yates, Benavides, and Gonźalez (2006) presented a framework for the automatic identification of user interests, based on the analysis of query logs. The researchers found that supervised learning could identify the user interests given certain established goals and categories. However, with unsupervised learning, one can validate the goals and categories used, refine them, and then select those most appropriate to the user's needs.

In addition to analyzing query logs, other researchers have focused on how to classify the huge transactional data set and cluster queries in order to use such records to explore user-system interactions. Some experts have investigated the process of how different search engines collect records as well as the applications of logging client-side interactions on the final analysis (Jansen, 2006 a, b; Özmutlu, et. al, 2004).

For example, Jansen (2006b) examined the characteristics and changes in Web searching from nine studies of five Web search engines from the perspectives of session length, query length, query complexity, and content viewed among the Web search engines. Another group of researchers tried to establish modules, such as searching quality measure and biased rank aggregation models to design a personalized Web search system. Fan (2006) proposed a novel representation scheme of nonlinear ranking function and compared this new design to the well-known Vector Space model. Fan then tested the new representation scheme with the GP-based discovery framework in a personalized search context using a TREC Web corpus.

However, this line of research is primarily descriptive and not supported by theoretical techniques. Nowadays, people are beginning to use more delicate methodologies to analyze the interactions between users and systems (Meghabghab & Kandel, 2004; Almpanidis, Kotropoulos & Pitas, 2007). In these efforts, researchers use statistical methods to analyze users' behaviors and to identify valuable information using transactional logs (Spink, Jansen, Wolfram, & Saracevic, 2002; Marable, 2003; Jansen & Spink, 2006). These efforts are usually empirical descriptions of past user behaviors.

One of the most challenging problems against building an efficient model of Web search is that search engine transactional logs contain technically discrete time series data. However, NNs are good tools for identifying relationships between inputs and outputs from a set of examples. Due to the NN approximation properties as well as the inherent adaptation features of these networks, NNs may have wide application for modeling of nonlinear systems (Giles, Lawrence & Tsoi, 2001). Unfortunately, only a few NNs have been applied to Web search engines.

Özmutlu, Çavdur, Spink, and Özmutlu (2002) provided an analysis of contextual information in esearch engine query logs. The study proposed a topic identification algorithm using artificial neural networks (ANNs). A sample from the Excite data log was selected to train the NNs, and then the NN was used to identify topic changes in the data log. The researchers reported that topics shifts were estimated correctly, with a 77.8 percent precision in the overall database.

In a follow-up of this research, Özmutlu, Spink, and Özmutlu (2004) provided the results from a comprehensive time-based Web study of US - based Excite and Norwegian-based Fast Web search logs, exploring variations in user searching related to the changes in time of the day. The researchers reported that the analysis of the datasets was very useful to Web search engines for reconstructing the search structure and reallocating the resources with respect to different periods.

However, neither of these studies provided an efficient model to both identify user-system interaction and reliably predict future user behaviors, especially user CTR, a primary metric in search engine evaluation for both organic and sponsored results.

To address this gap in current research, we construct a NN to study user-system interaction and to provide an efficient mechanism to predict user behaviors. In the following sections, we introduce two primary NNs and then apply each network method to train the data set in order to compare the fitting results of each approach. We follow this by conducting a sensitivity analysis of the input neurons based on the better fitted NN method, which is MLPN, in order to determine which types of information represented in the transactional log are predictive of users' click rates. We then present results, followed by discussion and implications for future work.

Chapter 3

Methodology

3.1 Neural Networks

Artificial Neural Networks (ANNs) are complex nonlinear distributed systems, and as a result, they have broad application. Many of NNs' remarkable properties result from their origins as biological information processing cells. NNs have two main application areas: open loop and closed loop feedback control.

In the open loop application, NNs are used for classification, or pattern recognition, and function approximation. In order to perform either of these functions, however, NNs must be trained and the widely used training technique of NNs is backpropagation error algorithm. This training technique involves a forward pass in order to compute responses corresponding to the input patterns followed by a backward pass to adjust the synaptic weights. Both passes are repeated until the actual responses of the network match the desired ones (Kampolis, Karangelos, and Giannakoglou, 2004). Feedforward networks are memory-less in the sense that their response to an input is independent of the previous network state.

Unlike open loop NNs, closed loop NNs are dynamic systems. When a new input pattern is presented, neuron outputs will be computed. Because of the feedback paths, the inputs to each neuron are modified, which leads the network to enter a new state. Consequently, different network architectures require different learning algorithms. For this project, however, we use open loop feedforward NNs.

Open loop feedforward NNs have been widely used in the area of system identification and function approximation. Since the purpose of this thesis is to explore the behaviors of online users and to discover which information shown in the transactional log influences and predicts the final CTR, we designed two primary open loop NNs and, after tuning the networks, analyzed the weights of each input element. Knowing how specific types of information impact CTR will allow commercial search engine companies to master the user-system interactive data to improve the CTR and design more efficient searching algorithms to increase the CTR.

For this research project, we use multilayer backpropagation neural network (MLPN) and radial basis function neural network (RBFN) to train the network and to perform the system identification.

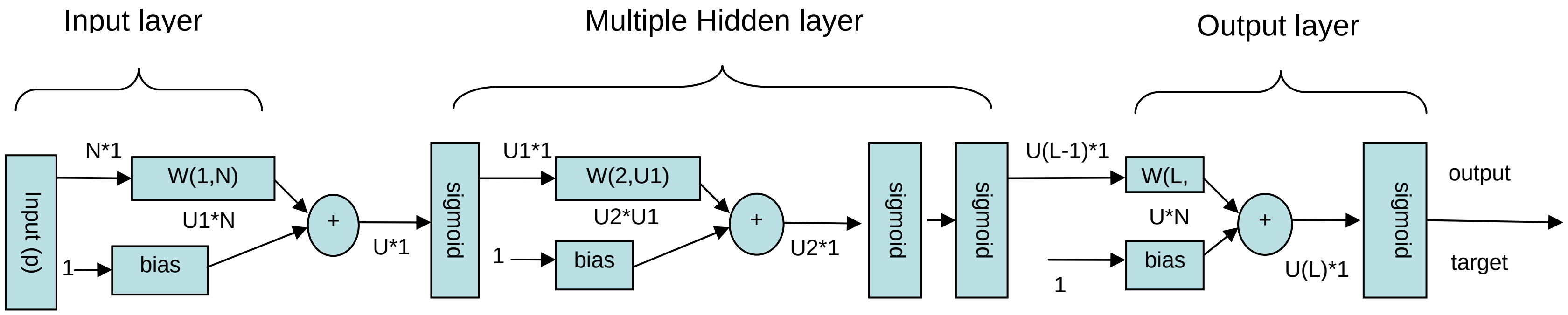
3.2 Multilayer Backpropagation Neural Network

MLPNs have multiple layers and use the backpropagation algorithm to tune the neuron's weights. Using a feedforward NN to process complex signals can be viewed as performing a curve-fitting operation in a multidimensional space. This may explain why most of the applications in this field employ NNs to realize some complex nonlinear decision functions. The theoretical justification for such applications is that, provided that the NN structure is sufficiently large, any continuous function can be approximated to within an arbitrary accuracy by carefully choosing parameters in the network. The key power provided by such networks is that they admit simple algorithms where the form of the nonlinearity can be learned from the training data. Thus, the models are extremely powerful, have strong theoretical properties, and apply well to a vast array of real-world circumstances.

Generally, the backpropagation algorithm in the multilayer feedforward network is sufficient for completing system identification and has wide application for other areas as well. The backpropagation training method is simple even for complex models with hundreds or thousands of parameters. The intuitive graphical representation and relatively simple designs of these networks enable designers to test different models quickly and easily; NNs are thus a flexible heuristic technique for performing statistical pattern recognition with complicated models. The conceptual and algorithmic simplicity of backpropagation, along with its success on many real-world problems, make it a mainstay in adaptive pattern recognition. In this section, we introduce its basic structure and provide the pseudo-code to train the transactional log.

3.2.1 Basic Structure

MLPNs often have one or more hidden layers followed by an output layer of linear neurons. Multiple layers of neurons with nonlinear transfer functions (i.e. sigmoid nonlinearity function) allow the network to learn nonlinear as well as linear relationships between input and output vectors. Generally, such networks are more efficiently trained with standardized data. In this thesis, we use normalized input and target data as the training and testing sample, and use sigmoid transfer function as the activation function to constrain the output from hidden layers of the network within the range from 0 to 1. There is no clear way of determining how many hidden neurons and layers are necessary to form a decision region sufficiently complex to satisfy the demands of a given problem. Thus, parameters required are best determined based on experiments. For the current project, we designed a NN with a flexible number of layers so in order to filter out nonlinear relationships as much as possible. After building the structure of network, we also designed a learning algorithm to fit the desired output. Figure 3-1 shows the detailed structure of an MLPN in detail.



Note:
N — number of input elements
U(i) – number of units in layer i
W(i,j) — weight vector for layer i which has j units

Figure 3-1: MLPN structure

3.2.2 Training Algorithm

Training the data set for MLPNs consists of two parts: forwarding the network and backpropagating. Forwarding the network means that all outputs are computed using sigmoid thresholds of the inner product of the corresponding weight and input vectors. Backpropagating the network entails transmitting errors backwards through the network by apportioning them to each unit according to the portion of the error for which the units are responsible. In this thesis, we use DELTA backpropogation method to train the NN. Because numerous textbooks and papers have illustrated the basic algorithms of backpropagating the network (see for example Haykin, 1999)., here we simply list the notation of variables and algorithms used to train the network.

<u>Variable Notation:</u>

$\overset{\rho}{x}{}^{l}_{j}$: Input vector for unit j in layer l. The input from unit i in layer l to unit j in layer l+1 could be denoted by x^{l+1}_{ji} .

$\overset{\varpi}{w}_{j}$: Weight vector for unit j in layer l. The weight between unit i in layer l and unit j in layer l+1 could be denoted by w^{l+1}_{ji} . In addition, we use $w^{adj^{l}}{}_{ji}\left(t\right)$ to represent the adjusted weight at the tth iteration.

$\overset{\varpi}{z}_{j}$: Weighted sum of the inputs for unit j in layer l.

$\overset{\varpi}{o}_{j}$: Output vector for unit j in layer l.

$\overset{\omega}{t}_{j}$: Target vector for unit j in the output layer.

η : Learning rate, in this study, η=0.25.

n_{l} : Number of units in layer l.

Bias: The bias for threshold function in each layer.

α : Momentum, which means the proportion of previous adjusted weight needed to adjust the current weight for the whole NN. In order to increase the learning rate without leading to oscillations, Rumelhart and Colleagues (1986) suggested a modification to generalized delta to include a momentum term. In our study, α =0.9.

δ_j^l : First partial derivative of sum square error w.r.t the input of each unit, $\delta_j^l = \frac{\partial E}{\partial z_j^l} = -(t_j^l - o_j^l)(1 - o_j^l)o_j^l$.

Gain: Proportion of δ needed to tune the NN.

Data Set:

We have three data sets to construct the NN:

1. Training sample: $\langle \overset{\varpi}{x}_j, \overset{\varpi}{t}_j \rangle$.
2. Testing sample: $\langle \overset{\varpi}{x}_j^{l}, \overset{\varpi}{t}_j^{l} \rangle$.
3. Evaluating sample: $\langle \overset{\varpi}{x}_j^{\prime l}, \overset{\varpi}{t}_j^{\prime l} \rangle$.

Training algorithm:

1. Normalize the input and target value into the range of [LO, UP], i.e. [0.1, 0.9].
2. Generate a feedforward network through all the layers (see figure 3-1):

   (1) Input the instance $\overset{\varpi}{x}_j$, and calculate the weighted sum of inputs and weights $\overset{\varpi}{z}_j = \overset{\varpi}{w}_j \cdot \overset{\varpi}{x}_j$.

(2) Put the weighted sum into the sigmoid activation function, and get the output from each layer: $\overset{\varpi}{o}_j = \frac{1}{1+e^{gain\times z_j^l}}$. Regard the output of layer l as the input of layer l+1.

3. Initialize all the weights to small random values (e.g. between -0.5 and 0.5).

4. Use DELTA backpropagation method to train the network backwards using calculated error between target and output in each layer until the termination condition is met.

   For each training sample $\left\langle \overset{\varpi}{x}_j, \overset{\varpi}{t}_j \right\rangle$, which could be randomly picked through the training data set:

   (1) Based on the feedforward NN constructed in step 2, calculate the error of the output layer units $\delta_j^l = gain\times(t_j^l - o_j^l)(1-o_j^l)o_j^l$. At the same time, we can calculate the train error of the whole NN as $\sum_j gain\times(t_j^l - o_j^l)(1-o_j^l)o_j^l$.

   (2) Calculate the error of the units in each hidden layer $\delta_j^l = gain\times o_j^l(1-o_j^l)\sum w_{ji}^l\delta_i^{l+1}$.

   (3) Update NN weight w_{ji}^l for each unit i at each layer l as follows: $w_{ji}^l(n) = w_{ji}^l(n) + \eta\delta_j^l o_i^{l-1} + \alpha {w^{adj}}^l_{ji}(n-1)$, here $w_{ji}^l(n)$ means the weight connecting neuron i and neuron j at the nth iteration and ${w^{adj}}^l_{ji}(n)$ means the adjusted weight at the (n-1)th iteration.

For each testing sample $\left\langle \overset{\varpi}{x}_j^{\prime l}, \overset{\varpi}{t}_j^{\prime l} \right\rangle$,

(1) Use the constructed network and testing sample to calculate the output error for the entire testing sample:

testing error= $\sum_j gain \times (t_j^{\prime l} - o_j^{\prime l})\left(1 - o_j^{\prime l}\right)o_j^{\prime l}$.

(2) Use the minimum testing error variable to restore the minimum testing error found at each iteration.

Termination condition:

If testing error is less than β *minimum testing error (i.e. β =1.2), terminate the training process.

5. For each evaluating sample $\left\langle \overset{\varpi}{x}_j^{\prime\prime l}, \overset{\varpi}{t}_j^{\prime\prime l} \right\rangle$, evaluate the network.

3.3 Radial Basis Function Neural Network

The RBFN is an alternative to highly nonlinear-in-the-parameters NNs, which means the determinants of neural centers have high nonlinearity. Traditionally, the RBF method has been used for strict interpolation in multidimensional space. The original RBF method requires that there be as many RBF centers as data points.

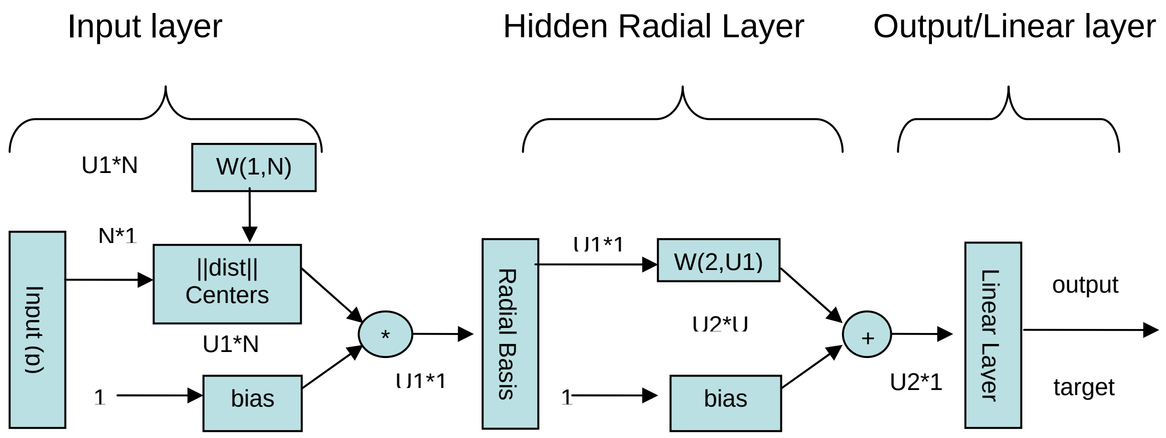
By fixing all RBF centers and nonlinearities in the hidden layer, an RBFN can be regarded as a special two-layer network with linear parameters. Thus, the hidden layer performs a fixed nonlinear transformation with no adjustable parameters and maps the input space onto a new space. The output layer then implements a linear combiner on this new space with the only adjustable parameters being the weights of this linear combiner.

These parameters, then, can be determined using the linear least squares (LS) method, which is an important advantage of this approach (Chen, et al. 1991).

The increasing popularity of the RBFN is due to their simple topological structure, their locally tuned neurons, and their ability to learn fast. The RBFN is a multivariate interpolation method, which rivals the MLPN in efficiency (Park and Sandberg, 1991). RBFNs can require more neurons than standard feedforward backpropagation networks, but they can often be designed in a fraction of the time it takes to train standard feedforward networks. In this section, we introduce its basic structure and provide the pseudo-code to train the transactional log.

3.3.1 Basic Structure

RBFNs usually consist of two layers in addition to the input layer: a hidden radial basis layer with U1 units (or centers) and an output linear layer with U2 units. Over the hidden layer units, a nonlinear mapping from an input space to a hidden one is carried out, while the mapping from the hidden layer to the output layer is linear. Each unit in the hidden layer employs a radial basis function, such as a Gaussian kernel, as the activation function. The radial basis function is centered at the point specified by the weight vector associated with the units. Both the positions and the widths of these kernels must be learned from the training patterns. Each output unit implements a linear combination of these radial basis functions. With respect to the function approximation, the hidden units provide a set of functions that constitute a basic set for representing the input patterns in the space spanned by the hidden units. The basic structure of RBFN is shown in Figure 3-2.



Note:
N—number of input elements
U1 –number of units/centers in hidden radial basis layer
U2 –number of units/centers in linear layer
W(i,j)—weight vector for layer i which has j units

Figure 3-2: RBFN structure

3.3.2 Training Algorithm

A variety of learning algorithms exist for training the RBFN. The basic one employs a two-step learning strategy or a hybrid learning strategy. It estimates the kernel positions and the kernel widths by using an unsupervised clustering algorithm, followed by a supervised least mean square (LMS) algorithm to determine the connection weights between the hidden layer and the output layer. In our study, we use K-means clustering method to do the clustering of the hidden layer centers. Given that the output units are linear, a non-iterative algorithm can be used. After an initial solution is obtained, a supervised gradient based algorithm is used to refine the network parameters. From the perspective of processing the network, the training process with the RBFN consists of two primary processes: forward the network and backward the network. Since the basic idea of forward the network is almost the same as that of the MLPN, here, we list the differences between the MLPN and the RBFN. Following this list, e e introduce the primary learning algorithm we used for this project, a combination of the K-means clustering method and the supervised least mean square algorithm.

Differences between the MLPN and the RBFN (Haykin, 1999):

1. An MLPN may have one or more hidden layers, whereas an RBFN (in its most basic form) has a single hidden layer.
2. Typically, the computation neurons of an MLPN, located in a hidden or output layer, share a common neuronal model. On the other hand, the computation neurons in the hidden layer of an RBFN are quite different and serve a different purpose from those in the output layer of the network.

3. The hidden and output layers of an MLPN used as a pattern classifier are usually all nonlinear, but the hidden layer of an RBFN is nonlinear, whereas the output layer is linear.
4. The activation function of each hidden unit in a MLPN computes the inner product of the input vector and the synaptic weight vector of that unit. The argument of the activation function of each hidden unit in an RBFN, however, computes the Euclidean norm (distance) between the input vector and the center of that unit.
5. The MLPN constructs global approximations to the nonlinear input-output mapping. On the other hand, RBFNs using exponentially decaying localized nonlinearities (e.g. Gaussian functions) construct a local approximation to the nonlinear input-output mappings.

Variable Notation:

$\vec{c}_j(n)$: j^th^ center at the n^th^ iteration.

N : Number of centers.

$\vec{x}_j$: Input vector for unit j in the input layer.

$k(\vec{x})$: Index of the best-matching (winning) center for the input vector $\vec{x}$.

$\vec{w}_j$: Weight vector for center j in the radial basis layer and the weight between unit j in the radial basis layer and unit i in the output layer could be denoted by w_{ji}. Moreover, we use $w^{adj}{}_{ji}(m)$ to represent the adjusted weight at the m^th^ iteration. In this thesis, the output layer has only one unit, which means j=1. Therefore, we could also use w_j and $w^{adj}{}_{j}(m)$ for short. Likewise, the notations below could be simplified in the same way.

$\overset{\varpi}{z}_j$: Weighted sum of the input into the output layer.

$\overset{\varpi}{o}{}_j^R$: Output vector for unit j in the radial basis layer.

$\overset{\varpi}{o}{}_j^o$: Output vector for unit j in the output layer.

$\overset{\omega}{t}_j$: Target vector for unit j in the output layer.

η_1 : Learning rate for the K-means clustering algorithm (in this study, η_1 =0.9).

η_2 : Learning rate for backpropagation algorithm (in this study, η_2 =0.25).

u^R : Number of units in the radial basis layer.

u^o : Number of units in the output layer, in this study, u^o =1.

Bias: Bias for the threshold function in each layer.

α : Momentum (in this study, α =0.9).

δ_j : First partial derivative of the sum square error w.r.t the input of each unit,

$$\delta_j = \frac{\partial E}{\partial z_j} = -(\overset{\varpi}{t}_j - \overset{\varpi}{o}_j)\left(1 - \overset{\varpi}{o}_j\right)\overset{\varpi}{o}_j .$$

Gain: Proportion of δ needed to tune the NN.

Training Algorithm--procedure:

1. Normalize the input and target value into a range of [LO, UP], i.e. [0.1, 0.9].

2. Use K-means algorithm to generate U1 centers to cluster the samples:

   (1) Initialization: choose random values for the initial centers $\overset{\varpi}{c}_j(0)$; the only restriction is that these initial values should be different. It may also be desirable to keep the Euclidean norm of the centers small.

(2) Similarity Matching: find $k(\vec{x})$ at iteration n by using the minimum-distance Euclidean criterion: $k(\vec{x}) = \arg\min \left\| \vec{x}(n) - \vec{c}_j(n) \right\|$.

(3) Updating: adjust the centers of the radial basis functions, using the updating rule:

$$\vec{c}_j(n+1) = \begin{cases} \vec{c}_j(n) + \eta_1 \left[x(n) - \vec{c}_j(n) \right] & j = k(\vec{x}) \\ \vec{c}_j(n) & otherwise \end{cases}.$$

(4) Continuation: increase n by 1, go back to step 2 and continue the procedure until no noticeable changes are observed in the centers.

3. Generate a feedforward network through all the layers (see figure 3-2):

(1) Input the normalized instance $\vec{x}_j$, with given centers, calculate the distance between the input and the centers $\left\| \vec{x}_j - \vec{t}_j \right\| = \sqrt{\sum_i (x_j{}^i - t_j{}^i)}$, here upper note i means the element index of the vector.

(2) Put the distance $\left\| \vec{x}_j - \vec{t}_j \right\|$ into the radial basis activation function and get the output as: $\vec{o}_j^{R} = e^{-\left\| \vec{x}_j - \vec{t}_j \right\|^2}$ and regard the output of the radial basis layer as the input of the output layer**.**

(3) Calculate the output of the linear output layer as $\vec{o}_j^{o} = \frac{\sum_i w_{ji} \vec{o}_i^{R}}{N}$.

4. Initialize all the weights to small random values (e.g. between -0.5 and 0.5).

5. Use backpropagation method to train the network backward using calculated error between the target and the output in each layer until the termination condition is met.

For each training sample $\langle \vec{x}_j, \vec{t}_j \rangle$, which could be randomly picked from the whole training set:

(1) Based on the feedforward network constructed in step 3, calculate the error of the output layer unit $\delta_j = gain \times (\vec{t}_j - \vec{o}^{\,o}_j)(1 - \vec{o}^{\,o}_j)\vec{o}^{\,o}_j$. At the same time, we calculate the training error of the NN as $\sum_j gain \times (\vec{t}_j - \vec{o}^{\,o}_j)(1 - \vec{o}^{\,o}_j)\vec{o}^{\,o}_j$.

(2) Update each network weight w_{ji} for each unit between the radial basis layer and the output layer as follows: $w_{ji}(n) = w_{ji}(n) + \eta\delta_j o^o_j + \alpha w^{adj}{}_{ji}(n-1)$, here $w_{ji}(n)$ means the weight at the nth iteration and $w^{adj}{}_{ji}(n)$ means the adjusted weight at the (n-1)th iteration.

For each testing sample $\langle \vec{x}'_j, \vec{t}'_j \rangle$,

(1) Use the established network and testing sample to calculate the output error for the entire testing sample:

testing error=$\sum_j gain \times (t'_j - o'^o_j)(1 - o'^o_j)o'^o_j$.

(2) Use the minimum testing error variable to restore the minimum testing error found at each iteration.

Termination condition:

If testing error is less than β*minimum testing error (i.e. β=1.2), terminate the training process.

6. For each evaluating sample $\langle \vec{x}_j, \vec{t}_j \rangle$, evaluate the NN.

Chapter 4

Experimentation and Analysis

4.1 Basic Data Analysis

According to Pierre (2000), 1.5 million users will log on to different search engines and submit information retrieval requests every day, and transactional logs of this size will surely be problematic to analyze.

In this thesis, we have two sets of databases: one is a daily data set, the Dogpile search engine transactional log; the other one is a weekly data set, the Vivisimo search engine transactional log. The Dogpile transactional log contains 4,193,956 pieces of records from May 15th, 2006. The Vivisimo transactional log contains 609,113 pieces of records collected from March 28th, 2006 to April 4th, 2007. Dogpile and Vivisimo are two popular meta-search engines. As the name implies, meta-search engines search in a number of search engines, usually less than a dozen or so, and return the top 10 websites from each search engine that relate to the search criteria. The results for a search using a meta-search engine will be much lower in number than the same search performed using a keyword search engine. The time limits may prevent some individual engines from being included in any search performed, so the results may vary from search to search and from day to day, even though users may be typing the search exactly the same way each time.

For this study, we conducted both basic and extended data analysis. In the basic data analysis part, we first explain the information represented in the transactional log explicitly and then present the information patterns shown in the transactional log based on the time series theory. Table 4-1 shows the fields/information included in these logs.

Table 4-1: Fields in the transactional log

Field	Description
Record Number	The number of records representing a single user.
IP Address	The IP address recording the computer on which the searchers log on.
Cookie	Parcels of text sent by a server to a Web browser and then sent back unchanged by the browser each time it accesses that server. Cookies are used for authenticating, tracking, and maintaining specific information about users, such as site preferences and the contents of their electronic shopping carts.
Time	The time when the interaction was recorded by the search engine server.
Query	The terms of the queries that the user typed into the search engine text box when searching for information.
Vertical	There are five types of vertical (Web, Audio, Image, Video, News) representing different content collections. They are automatically divided by the search engines and provide a convenience for the searchers to find different information in different formats.
Sponsored	One of two possible types of links retrieved and presented on the SERP. Sponsored links appear because a company, organization, or individual purchased the keywords that the searchers used in the search query. If a searcher entered a sponsored link, then this record will show 1.
Organic	The other type of link retrieved and presented on the SERP. These links are retrieved by search engine using its proprietary matching algorithm. If the searcher entered an organic link, then this record will show 1.
Browser	The browser used by the searchers.
Location	The place/country where a user used the search engine.

We also calculated three additional attributes for each record, presented in Table 4-2.

Table 4-2: Additional calculated fields in the transactional log

Field	Description
User Intent	There are three categories of user intent, which are informational, transactional, and navigational. They reflect the information type retrieved by the search engine. For this process, we select a sample of records containing not only the query but also other attributes, such as the order of the query in the session, query length, result page, and vertical, and then manually classified the queries in one of three categories, which is derived from work in Rose & Levinson (2004).
Query length	The number of terms contained in a particular query.
Results Pages	A number representing the SERP viewed (blank is first page, 1 is second page, etc.).

In the basic analysis, we wanted to get a straightforward idea of online users' behaviors based on a time series analysis. We first use Matlab connected to the SQL Server to perform the statistical calculation based on fields of information type, rank of SERP, type of vertical, type of link (sponsored or organic), query length, and the page number that the clicked link belongs to. Based on the time series analysis theory, we had to ensure that each time unit we analyzed had equal time buckets so that statistical data would not be affected by the length of the time slot. For this thesis, we divided 4,193,956 pieces of records from Dogpile into 1080 equidistance groups and divided 609,113 pieces of records from Vivisimo into 7×600 groups. The basic statistical analyses for these two data sets follow.

For the Dogpile daily transactional log, data is based on a 24-hour daily transactional log (from 00:00 to 24:00). Figure 4-1 shows that the number of logs in records within each group, namely the population flow, goes up during the daytime and drops during the night. If we regard the extreme data that occurred at about the 70th and 270th buckets as aberrant or resulting from abnormal behaviors, such as some people continuously and maliciously logging in and out of a search engine within a short period, then we get the same results as other studies. According to the time based analysis on the Excite and Fast search engine studied by S. Özmutlu et al. (2004), the decrease in the queries per session indicates that Web search engine users might spend less effort on retrieving their information needs later in the day.

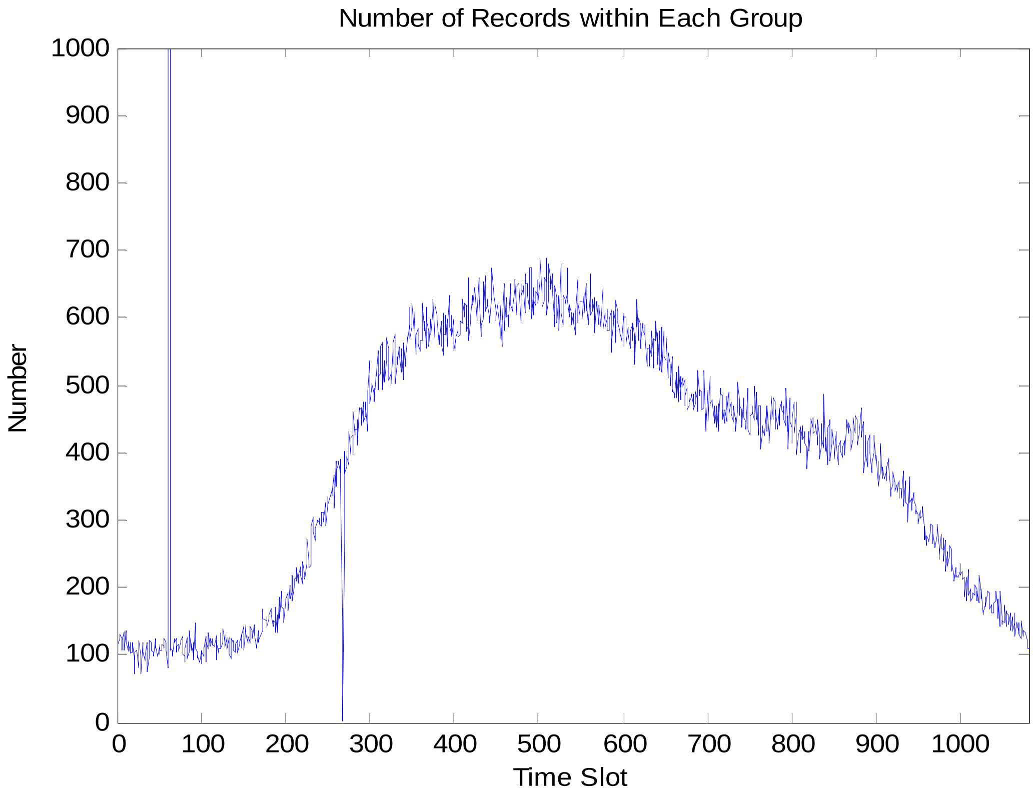


Figure 4-1: Number of records within each time slot for daily data set (Population Flow)

We can also analyze the popularization of different browsers by calculating the number of browsers used between each time slot. From Figure 4-2, we can see that most people prefer to use Internet Explorer (IE) browser compared with Firefox, Mozilla, and other browsers, and furthermore, the rate of use for Firefox, Mozilla and other browsers is rarely affected by time.

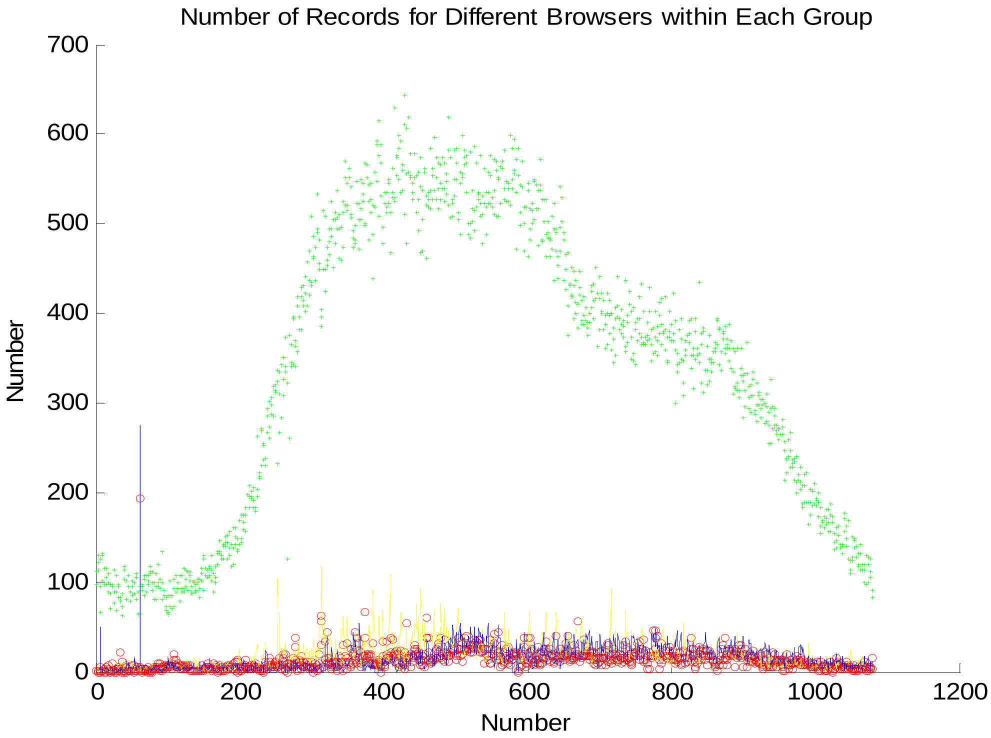


Figure 4-2: Number of records for different types of browsers within each time slot for daily data set (blue—Firefox, green—MSIE, red—Mozilla, yellow—other browsers)

Similarly, Figure 4-3 shows our analysis of the types of verticals searched by the users. People seem to prefer to use Web based information rather than images and audios.

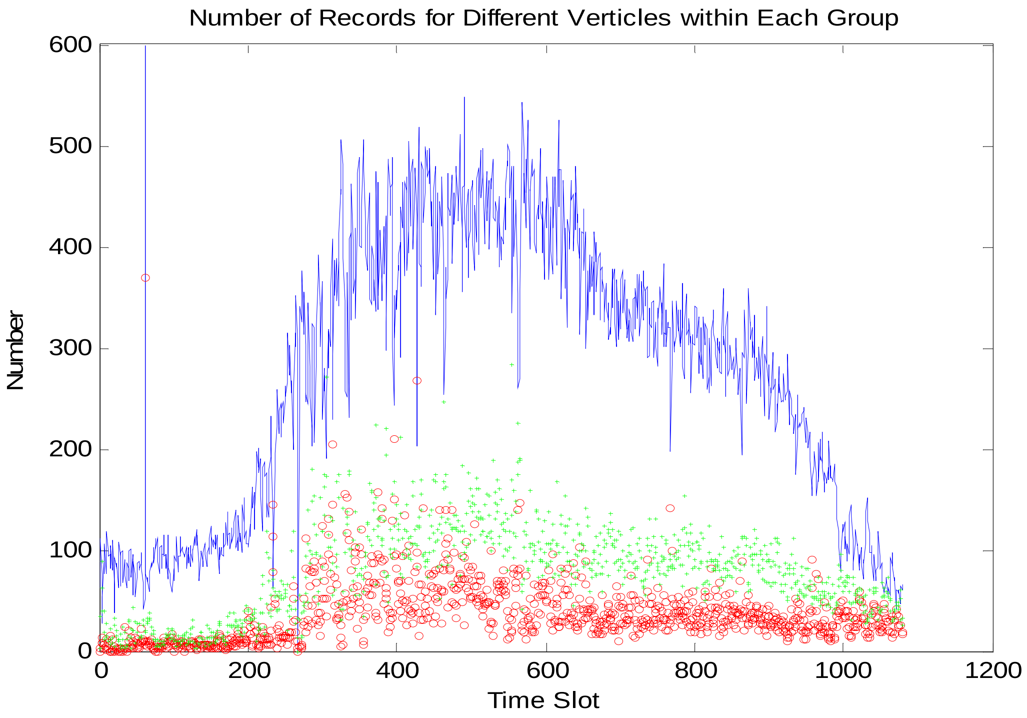


Figure 4-3: Number of records for different types of vertical within each time slot for daily data set (blue—Web, green—image, red--audio)

Figure 4-4 shows the condition for the types of information levels, namely which type of information searchers are looking for during the different periods. Most of the Web pages opened belong to the informational level. Pages containing transactional and navigational information take represent relatively small proportion of searches.

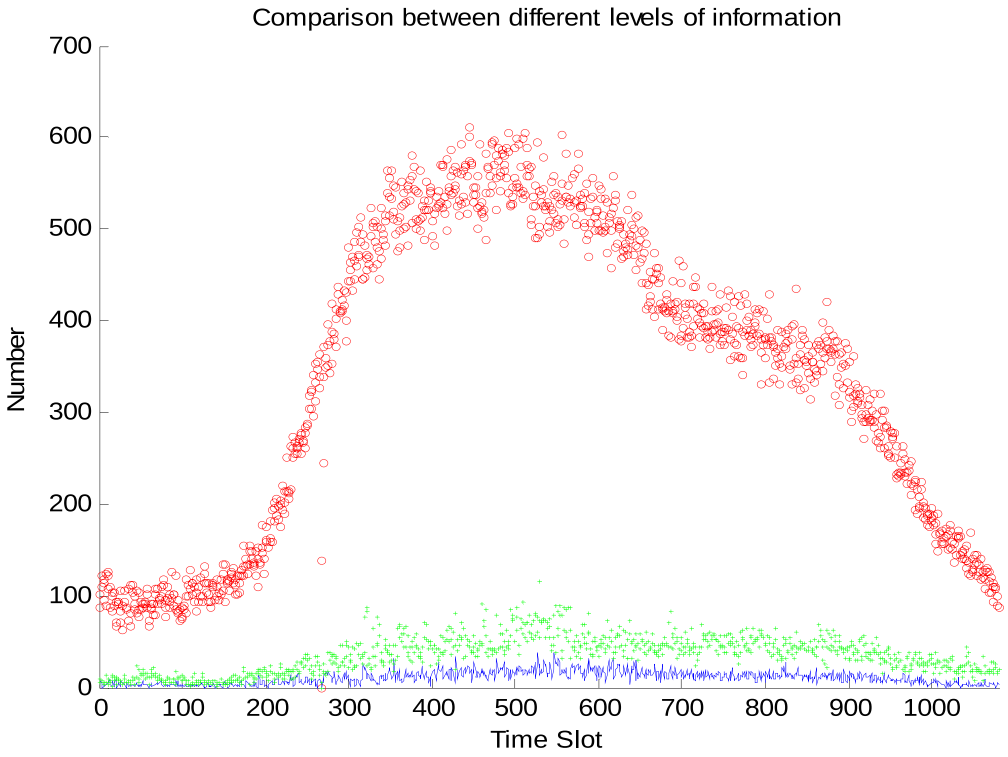


Figure 4-4: Number of records for different types of information level within each time slot for daily data set (blue—navigational, green—transactional, red—informational)

For Vivisimo's weekly transactional log, data is based on the weekly transitional log (from Monday to Sunday). Figure 4-5 shows the same trend for Dogpile's daily data. The click rate increases during the daytime and decreases during the night. Additionally, during the weekend, the use of search engines drops sharply (from time slot 3000 to 3600).

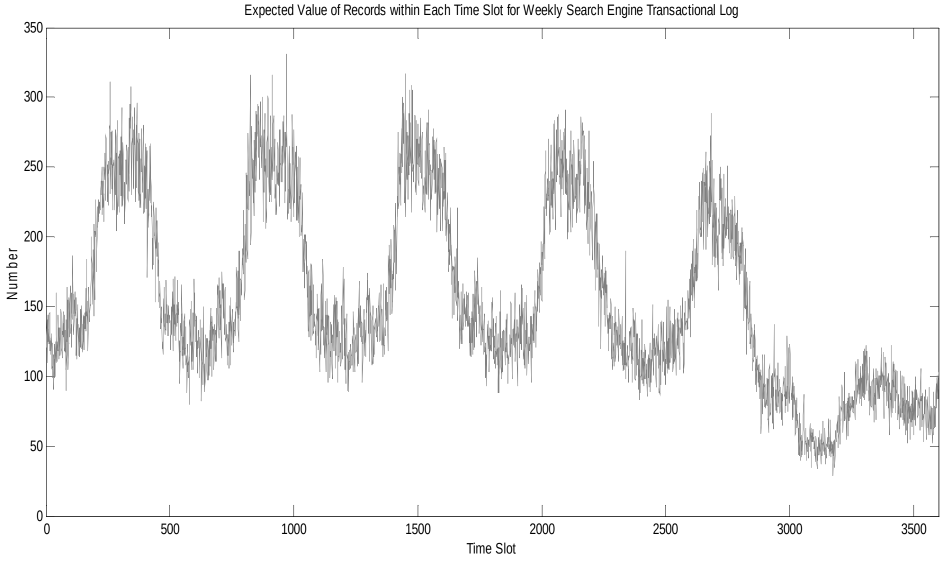


Figure 4-5: Number of records within each time slot (population flow) for weekly data set

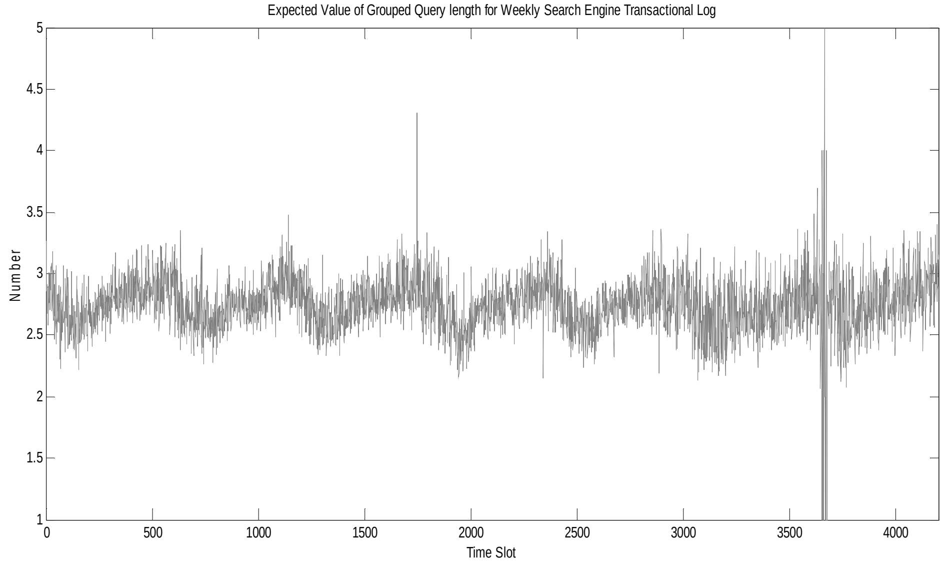


Figure 4-6: Average length of queries within each group for weekly data set

Figure 4-6 shows the expected value of grouped query length for weekly data. The trend shows as a stationary time series with a little seasonal trend. The average query length is about 2.8, which is almost the same as the 2.9 reported by Özmutlu (2004), and the 2.4 reported by Spink (Spink, et al., 2002).

In conclusion, we have a basic idea of how transactional logs show the behaviors of the online users and how to use transactional logs to retrieve the basic information. We know that users spend less effort on retrieving their information needs later in the day and during the weekend. Most people prefer using the IE browser, opening the informational result links, and accessing Web based information. These descriptive tidbits, however, cannot explain the potential interactions between online users and search engines. Moreover, the results cannot generate an efficient method to explore the potential for improving the efficiency and accuracy of search engines. To address such shortcomings of basic analysis, we use a more complicated quantitative analysis method to perform system identification and to discover what information influences the CTR, an important metric of the searching performance.

4.2 Extended Analysis—Neural Network Analysis

In this section, we use NNs to do the system identification, based on the transactional log data, in order to detect the significant factors influencing final CTRs. Since the Dogpile search engine transactional log is a complete daily data set, while Vivisimo's weekly transactional log is a sample weekly data set, we only use Dogpile's data set for the following on-line CTR analysis.

To begin the on-line CTR analysis, we do some basic indexing and calculation based on the given record and then select several potential inputs for the NN. Since CTR is based on each online user, we group the records according each unique IP address and

Cookie to determine a single user. Based on the records for each single user, we use the database SQL selection method to retrieve the information necessary to generate numerically formatted training, testing, and evaluating samples. Table 4-3 shows information arranged to train the NNs.

Table 4-3: Additional calculated data for training the network

Field	Description
Number of Records	The number of records representing a single user. SQL: select count(*) from <table name> group by IP, cookie
Average Query Length	Average query length typed by a single user. SQL: select avg(query_length) from <table name> group by ip, cookie
Rate of User Intent	We calculate the total number of users for each user intent (informational, transactional and navigational) and then calculate the rate by dividing the number of records for each user intent by the total number of users. SQL: select first(user_intent) into <new table name> from <table name> group by ip, cookie select count(*) from <new table name> group by user_intent
Rate of Browser	We calculate the total number of users for each browser (Firefox, Mozilla, MSIE, etc) and then calculate the rate by dividing the number of records for each browser by the total number of users. SQL: select distinct first(browser) from <table name> group by ip, cookie select first(browser) into <new table name> from <table name> group by ip, cookie select count(*) from <new table name> group by browser
Rate of Vertical Type	We calculate the total number of users for each vertical type (Web, Audio, News, Images, etc) and then calculate the rate by dividing the number of records for each vertical type by the total number of users. SQL: select distinct first(vertical_type) from <table name> group by IP, cookie select first(vertical_type) into <new table name> from <table name> group by IP, cookie select count(*) from <new table name> group by vertical_type
Mean Organic	We calculate the total number of records representing opened organic

Field	Description
Number	links and then calculate the rate by dividing the number of records for each opened organic links by the total number of records for each user. SQL: select sum(organic) from <table name> group by ip, cookie having organic <>’Null’ select count(*) from <table name> group by ip, cookie having organic <>’Null’
Average Opened Non-First Pages	We calculate the total number of records representing opened non-first pages and then calculate the rate by dividing number of the record for each opened non-first pages by the total number of records for each user. SQL: select sum(qtot) from <table name> group by ip, cookie having qtot <>’Null’ select count(*) from <table name> group by ip, cookie having qtot <>’Null’
Average Rank	The average rank of the links opened by each user. SQL: select avg(rank) from <table name> group by ip, cookie having rank <>’Null’
Non-clicked Number	The pages retrieved by the search engine but have not been opened by the users. SQL: select count(*) from <table name> group by ip, cookie having rank ==’Null’
Reformulation Rate	The times when the user changes the queries. SQL: select first(query) as unique_query, ip, cookie into <new table name> from <table name> group by ip, cookie, query select count(unique_query) from <new table name> group by ip, cookie
Log-in Time	The log in time for each user. SQL: select min(time) from <table name>
Log-out Time	The log out time for each user. SQL: select max(time) from <table name>
Time Range	The time frame spent by each user = log out time – log in time.
CTR	CTR = alpha * (reformulation rate / number of record) + beta*(Non-clicked Number / number of record) + gamma * (average rank), where alpha, beta and gamma are decimal fractional factors between 0 and 1. In this study, alpha=1, beta=0.5, gamma=0.01.

We group the records according to each unique IP address and cookie to determine a single user. For the Dogpile data set, we have information on 16,383 different users. However, this huge data set is not necessary to train the NNs because the principle of training is about how to use insufficient data to get necessary information. If we use all the records to do the training process, the construction of NNs will be meaningless.

Therefore, a smaller randomly selected data subset is appropriate to determine the characteristics and performance of the NNs while training the transactional log data set. Considering the computational time and efficiency of the network, we used 550 randomly selected user sessions as the training, testing, and evaluating data. We used the first 300 sessions as the training sample, and the next 200 sessions as the testing data, and the final 50 sessions as the evaluation sample. Using the sample, we determined which NN (MLPN or RBFN) best fit the curve for the significant analysis of the input neurons.

We then used the entire data set, divided into 29 groups, with each group containing 550 users' data, to complete the final analysis. First, we generated a random number for each user's data and then sorted the random numbers in increasing order. Based on the sorted random number, we divide 16,383 user records into 30 groups, with each group having 550 records, except the 30th group, which had 433 records. Since the last group was incomplete, we did not use it to tune the NN.

In this study, we selected nine kinds of information as the input to the NN, and the CTR will be regarded as the desired output (see Table 4-4).

Table 4-4: Inputs of the NN

Factor #	Factor Name	Description
Factor 1	Number of Record	The number of records representing a single user.
Factor 2	Sum of Rank	The total rank of links opened by each user.
Factor 3	Mean Organic Number	We calculate the total number of records representing the opened organic links and then calculate the rate by dividing the number of opened organic links by the total number of records for each user.
Factor 4	Mean Query Length	Average query length typed by a single user.
Factor 5	Rate of Browser	We calculate the total number of users for each browser (Firefox, Mozilla, MSIE, etc) and then calculate the rate by dividing the number of records for each user intent by the total number of users.
Factor 6	Rate of Vertical Type	We calculate the total number of users for each vertical type (Web, Audio, News, Images, etc) and then calculate the rate by dividing number of the records for each vertical type by the total number of users.
Factor 7	User Intent Rate	We first calculate the total number of users for each user intent (informational, transactional and navigational) and then calculate the rate by dividing number of records for each user intent by the total number of users.
Factor 8	Log in Time	The log in time for each user.
Factor 9	Time Range	The log out time for each user.

4.2.1 Comparison between the MLPN and the RBFN

Following the algorithm provided in the methodology section, we continuously trained the network until the termination condition, namely the current iteration's error for testing data set was greater than 1.2 times of the previous iteration's error, was satisfied. For the MLPN method, the parameters of the number of hidden layers and hidden neurons required were selected based on the experiments. After testing the network several times, we choose two hidden layers with four and six hidden neurons, respectively. In this thesis, one iteration means training the network using 3000 pieces of training data, which equals the number of epochs (10 in this study) times the number of records in the training data set (300 in this study). Using the java code shown in the appendix, we achieved the training error with respect to each iteration (see Figure 10 and Figure 11).

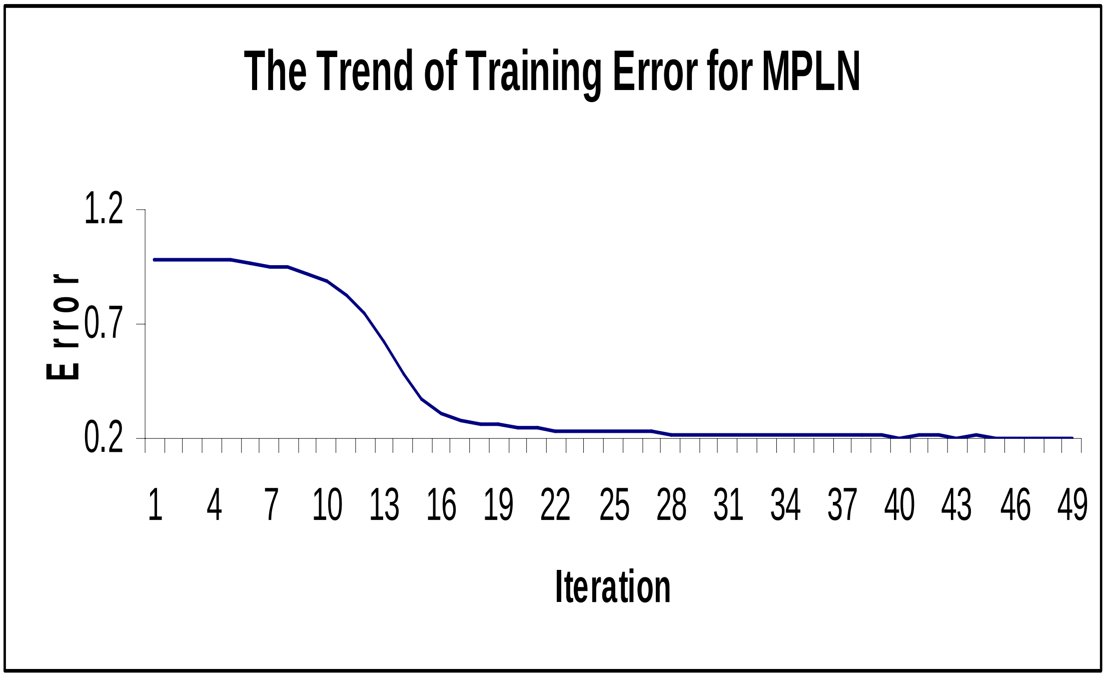


Figure 4-7: Training error for MLPN

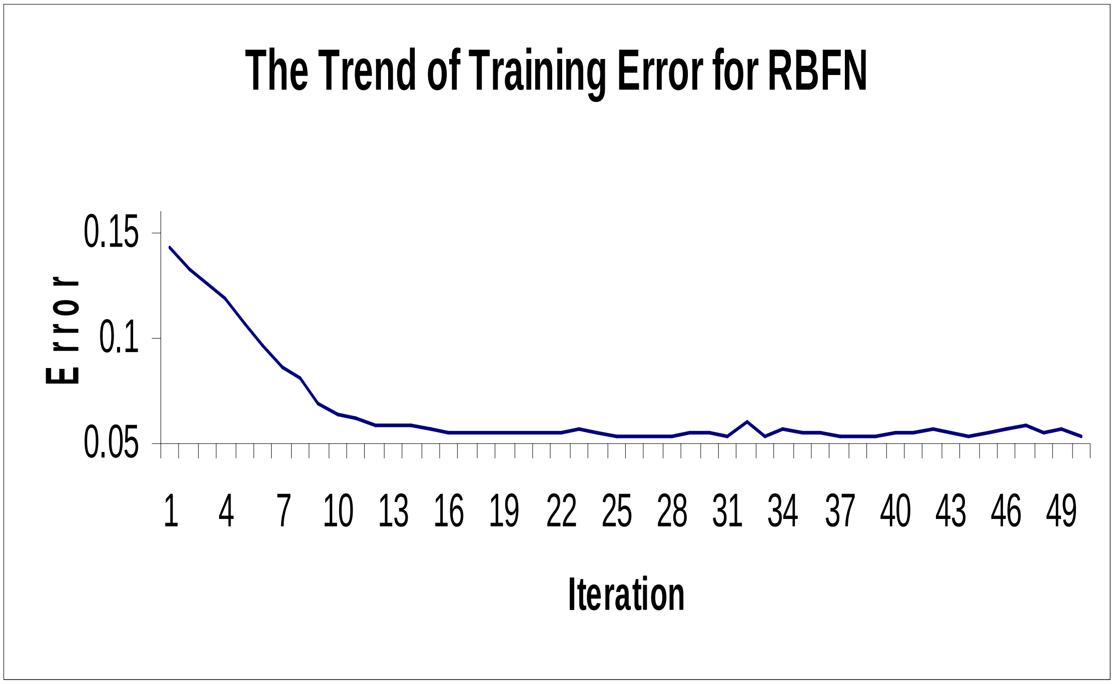


Figure 4-8: Training error for RBFN

Figure 4-7 and figure 4-8 show that the training error for the MLPN starts at about 0.9 and is close to 0.2 after iteration 29, while the training error for the RBFN starts at about 0.15 and shrinks almost to 0.05 after iteration 17. This phenomenon is explainable according to the training characteristics of the MLPN and the RBFN. The MLPN uses differentiable and continuous activation functions within hidden layers to screen out the nonlinear behaviors and to tune weights, while the RBFN uses linear output layer to tune the weights after ruling out all the nonlinear behaviors using clustered centers. Therefore, the RBFN deals with less random and irregular nonlinear data than the MLPN does. For this reason, the RBFN could begin to train the network with a lower training error and terminate the iteration earlier. Additionally, between iteration 7 and iteration 17, the training error for the MLPN drops dramatically, while the error does not change much for the other parts. As for the RBFN, the training error maintains the same slope.

Although the training error of the RBFN is much smaller than that of the MLPN, we could not say that the RBFN performs better than the MLPN because each calculates errors based on different input data sets. The error for the RBFN is based on the output coming out of the hidden layer, which has been screened of some nonlinear behaviors, thereby creating data that is more aggregated. We use the evaluation data set to test the fitting of the curve between the output data and objectives in order to see which NN worked better using the transactional log.

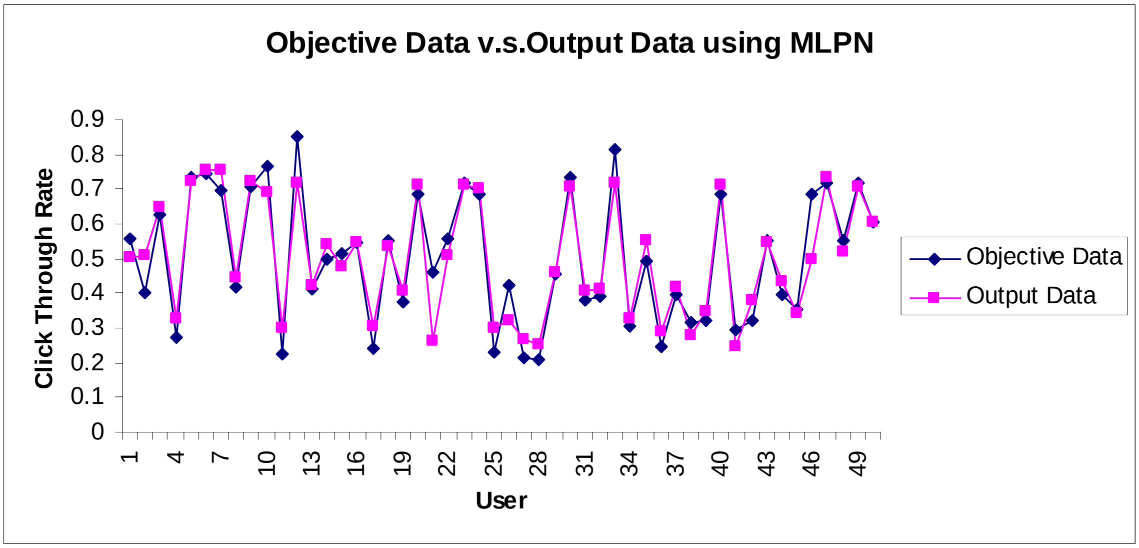


Figure 4-9: CTR Fitting Curve of sample data set using the MLPN

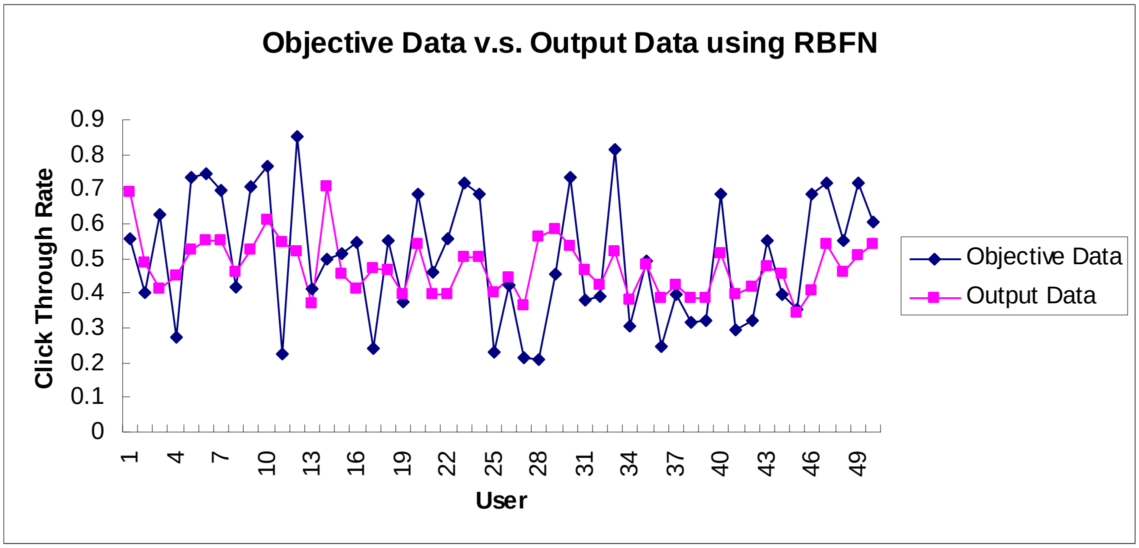


Figure 4-10: CTR Fitting Curve of sample data set using the RBFN

Figure 4-9 and Figure 4-10 show the fitting curves for the MLPN and the RBFN using the same evaluating data set. We can see that the MLPN performs much better than the RBFN in the fitting curves. In other words, the RBFN hidden layer cannot filter out the nonlinear behaviors as well as the MLPN hidden layer does. From a practical point of view, different users will have different searching styles, which is possibly the primary cause of high nonlinearity in the data set. Since the MLPN behaves much better than the RBFN does, in the rest of this study, we focus only on the sensitivity analysis of the input neurons based on the MLPN.

4.2.2 Sensitivity Analysis of the Input Neurons

Now we go back to figure 3-1, which represents the topology of the MLPN. As it is well known, the relative magnitudes of the weights decide the relative importance of inputs in determining the final output. Since the weights between the input layer and hidden layer keep changing when we tune the network, it is reasonable to say that the sum of weights from one element of the input vector to all the neurons in the hidden layer can represent the significant level of the impact of this input on the final output (CTR). Since the initial weights are randomly generated, we do the same analysis for 29 groups of user data and use the normalized average value to determine the final effects of each piece of input information.

Figure 4-11 shows the CTR fitting curve for all the 420,098 pieces of user data. We observe a good fitting curve between the objective data and output data generated by the MLPN.

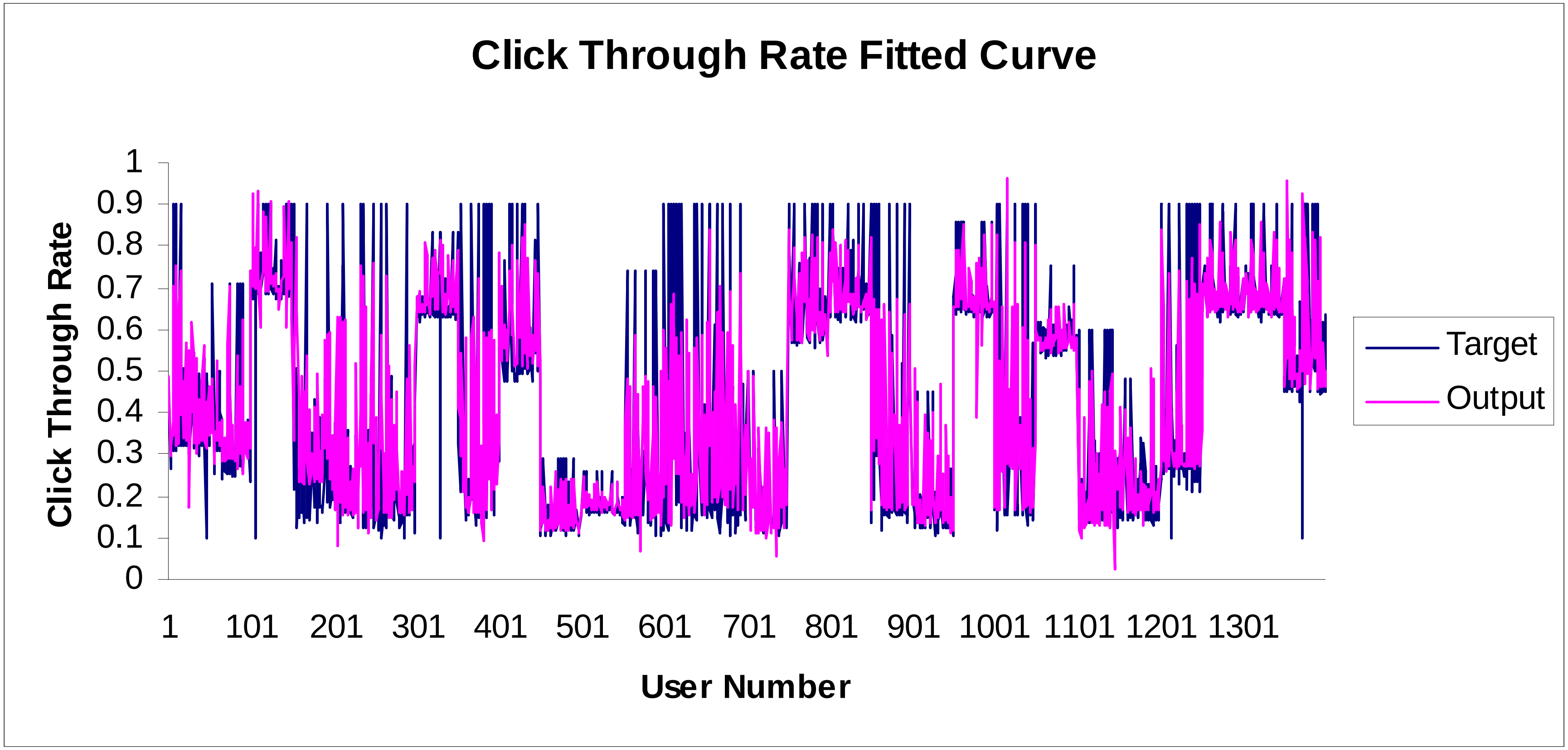
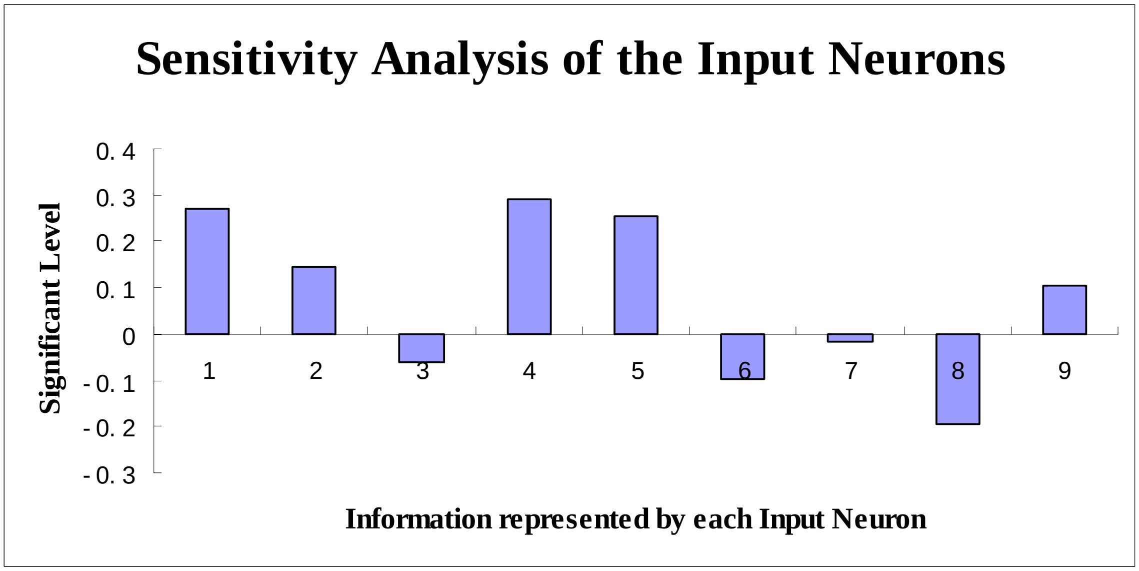


Figure 4-11: CTR Fitting Curve for the Whole Data Set



Note: 1-- Number of Record; 2-- Sum of Rank; 3-- Mean Organic Number; 4-- Mean Query Length; 5-- Rate of Browser; 6-- Rate of Vertical Type; 7-- User Intent Rate; 8-- Log in Time; 9-- Time Range

Figure 4-12: Sensitivity analysis of the input neurons with the MLPN

Figure 4-12 provides a clear indication of the impact on the final CTR for each input. Overall input 3 (Average opened organic links), input 6 (Vertical Type Rate), input 7 (User Intent Rate), and input 8 (First Log on Time) have negative effects, while input 1 (Number of Record), input 2 (Sum of Rank), input 4 (Average Query Length), input 5 (Browser Type Rate), and input 9 (Time Range) have positive effects. Additionally, input 3 (Mean Organic Number), input 6 (vertical type), and input 7 (user intent type) do not have much impact on the final CTR.

From a practical point of view, when a user frequently tries to reformulate the queries and open lots of links, the CTR will increase. The longer the query length typed in by the users for searching, the higher CTR will be. The CTR will not change much whether people are looking at organic or sponsored links. Interestingly, searchers using IE browsers may accept more results than those who use other browsers. Since input 8 (First

Log on Time) has negative effects, and input 9 (Time Range) has positive effects on the final CTR. CTR increases when the users stay longer or search earlier during the day.

Table 4-5 provides a clear idea of how each input of information impacts the CTR.

Table 4-5: Sensitivity analysis of the CTR

Factor #	Factor Name	Correlation with the CTR
Input 1	Number of Record	Has a positive correlation with the CTR. The CTR will increase while more user-logging-on information is recorded.
Input 2	Sum of Rank	Has a positive correlation with the CTR. The CTR will increase while the users open more links.
Input 3	Mean Organic Number	Does not have much impact on the final CTR.
Input 4	Mean Query Length	Has a positive correlation with the CTR. The CTR will increase while the users are trying to type in longer queries to do the searches.
Input 5	Rate of Browser	Has a positive correlation with the CTR. The CTR will go higher if the user uses MSIE other than other popular browsers.
Input 6	Rate of Vertical Type	Does not have much impact on the final CTR.
Input 7	User Intent Rate	Does not have much impact on the final CTR.
Input 8	Log in Time	Has a negative correlation with the CTR. The CTR will increase while the users do the searches in the early morning during the day.
Input 9	Time Range	Has a positive correlation with the CTR. The CTR will increase while the users stay longer to do the searches.

These findings are significant for commercial search engine companies who seek to increase the CTR. Search engines could provide more instructions and links during the evening hours so that users can focus more on the topics that interest them. Additionally, the search engine should focus more on the quality of search results on popular information types (i.e. informational) and vertical types (i.e. Web) and put less emphasis on other types in order to conserve searching costs (i.e. Transactional, News, etc). Since input 2 (sum of rank) has a positive correlation with the CTR, it is possible to increase the

CTR by rearranging the web page layout in order to make more links visible in a single page of meta-search engines.

Chapter 5

Conclusions and Future Research

In this thesis, we focused on how Web search engines' transactional logs can be very useful in examining system-user interactions using NN algorithms. In some sense, this study is a first step in using NNs in the analysis of human-system interactions for Web search data. This research explores the online behaviors of users so that commercial search engine companies can utilize the user-system interactive data to improve CTRs and design more efficient search algorithms.

The analysis in this thesis consists of two parts: the basic data analysis and the extended data analysis. The results of the basic analysis show that users will spend less effort on retrieving their information needs later in the day and during the weekend. Most people prefer using the Internet Explorer browser, clicking the informational result links, and accessing the Web based information. For the extended data analysis, based on the NNs methodologies, we designed two NNs (a RBFN and a MLPN), the characteristics and qualities of which are compared by screening out the nonlinear behaviors with reasonable explanations. The results show that the NNs perform well in handling large data sets, especially for data sets having huge unpredictable elements and abnormal behaviors. The MLPN, which proved to be more efficient for system identification using the transactional log, was used to detect the significant information on the final response — CTR. The results from the extended analysis show that the input 3 (Average Opened Organic Links), input 6 (Vertical Type Rate), input 7 (User Intent Rate), and input 8 e(First Log on Time) have negative effects, while input 1 (Number of Record), input 2 (Sum Rank), input 4 (Average Query Length), input 5 (Browser Type Rate), and input 9 (Time Range) have positive correlations with CTR.

From a practical point of view, the CTR increases when the users stay longer or conduct searches in early during the day. When a user frequently tries to reformulate the queries and open lots of links, the CTR will increase. The longer the query is typed in by the users for searching, the higher the CTR will be. However, whether people are looking at organic links or sponsored ones, the CTR will not change much. Additionally, it seems that the searchers using IE browsers may accept more results than those who use other browsers. According to these interesting findings, we could provide some suggestions to the commercial search engine companies for improving the performance of the searching algorithms.

This thesis is a first step in the field in which the search engine transactional log analysis uses NNs to study search data. However, there are still several limitations in present study. First of all, we did not consider different training algorithms for both the MLPN and the RBFN and did not analyze the best conditions by changing the initial parameters of the networks. Consequently, we cannot say the MLPN will always perform better than the RBFN in any situation. Secondly, in the extended part of this study part, the CTR as well as the selected influencing information is based only on the meta-searching transactional logs. The final results retrieved from our analysis may not represent the pattern shown by other types of search engines. Future work should combine the results from different search engine transactional logs to conduct a comprehensive study based on multiple types of resources.

References:

Almpanidis, G., Kotropoulos, C. & Pitas, I. (2007). Combining text and link analysis for focused crawling—An application for vertical search engines. Information Systems, 32, 886-908.

Baayen, R. H. 2001. Word frequency distributions, Text, Speech and Language Technology, Kluwer, Boston.

Beitzel, S.M,., Jensen, E.C., Chowdhury, A., Grossman, A., Frieder, O., Goharian, N. 2004. On Fusion of Effective Retrieval Strategies in the Same Information Retrieval System, Journal of the American Society for Information Science and Technology (JASIST), 55(10), 859-868.

Beitzel, S.M., Jensen, E.C., Lewis, D.D., Chowdhury, A., & Frieder, O. (2007). Automatic classification of Web queries using very large unlabeled query logs ACM Transactions on Information Systems, 25(2), Article No. 9.

Bruza, P.D., Dennis, S. 1997. Query reformulation on the internet: empirical data and the Hyperindex search engine. Proceedings of the RIAO 97 Conference, Montreal, 488-99.

Can, F., Nuray, R., & Sevdik, A. B. 2004. Automatic performance evaluation of Web search engines pages. Information Processing and Management, 40(3), 495–514.

Chen, S., Cowan, C. F. N. and Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. IEEE transactions on neural networks, 2(2), 302-309.

Chu, H., & Rosenthal, M. (1996). Search engines for the World Wide Web: a comparative study and evaluation methodology. In Proceedings of the 59th annual meeting of the American Society for Information Science, 127–135.

Cool, C., Belkin, N. J. and Koenemann, J. (1996). On the Potential Utility of Negative Relevance Feedback in Interactive Information Retrieval. In: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval 1996. p. 341. Available online.

Ding, W., & Marchionini, G. (1996). A comparative study of Web Search service performance. In Proceedings of the 59th annual meeting of the American Society for Information Science, Baltimore, MD, 136–142.

Eastman, C. M. 2002. 30,000 Hits may be better than 300: precision anomalies in Internet searches. Journal of the American Society for Information Science and Technology, 53(11), 879–882.

Eastman, C. and Jansen, B. J. 2003. Coverage, relevance, and ranking: the impact of query operators on Web search engine results. ACM Transactions on Information Systems. 21(4), 383 - 411.

Fan, W., Pathak, P., Wallace, L.2006. Nonlinear ranking function representations in genetic programming-based ranking discovery for personalized search. Decision Support Systems, 42 (3), 1338-1349

Giles, C.L., Lawrence, S., Tsoi, A.C. (2001). Noisy time series prediction using a recurrent neural network and grammatical inference, machine learning. 44 (1-2), 161-183.

Gordon, M., & Pathak, P. 1999. Finding information on the World Wide Web: the retrieval effectiveness of search engines. Information Processing and Management, 35(2), 141–180.

Haykin, S. 1999. Neural networks—a comprehensive foundation, second edition, Prentice Hall, 202-247.

Heckerman, D., Horvitz, E. (1998). In Inferring Informational Goals from Free-Text Queries: A Bayesian Approach. Paper presented at the Fourteenth Conference of Uncertainty in Artificial Intelligence, San Francisco, CA, USA. 230-237.

Hawking, D. (2006a, b). Web Search Engines: Part 1 and 2. IEEE Computer, 39(6, 8), 86-88, 88-90.

Ingwersen, P. (1996). Cognitive perspectives of information retrieval interaction: elements of a cognitive IR theory. Journal of Documentation, 52(1), 3-50.

Jansen, B. J. (2000). An investigation into the use of simple queries on Web IR systems. Information Research: An Electronic Journal, 6(1), 1–10.

Jansen, B. J., Resnick, M. (2006). An examination of searcher's perceptions of non-sponsored and sponsored links during ecommerce Web searching. Journal of the American society for information science and technology, 57(14), 1949-1961.

Jansen, B. J. and Molina, P. (2006). The effectiveness of Web search engines for retrieving relevant ecommerce links. Information Processing & Management. 42(4), 1075-1098.

Jansen, B. J., Spink, A. (2006). How are we searching the Would Wide Web? A comparison of nine search engine transaction logs. Information Processing and Management, 42 (1), 248-263.

Jansen, B. J., Booth, D., and Spink, A. (Forthcoming). Determining the informational, navigational, and transactional intent of Web queries. Information Processing & Management.

Marable, L. (2003). False oracles: Consumer reaction to learning the truth about how search engines work, results of an ethnographic study. Consumer WebWatch, 1-66.

Meghabghab, G., Kandel, A. (2004). Stochastic simulations of Web search engines: RBF versus second-order regression models. Information Sciences, 159, 1-28.

McCarthy, T. 2005. Yahoo! Goes to Hollywood. Time. 165(12), 50-53.

Neileson Netrating (2002). Top Web properties March 2002 [Website]. Nielsen/Netrating. Retrieved, 27 September, 2002. Available from http://www.nielsen-netratings.com.

Nicholson, S. (2000). Raising Reliability of Web Search Tool Research through Replication and Chaos Theory. Journal of the American Society for Information Science, 51(8), 724-729.

Özmutlu, S., Spink, A., & Özmutlu, H.C. 2004. A day in the life of web searching: an exploratory study. Information Processing and Management, 40(2), 319-345.

Park, J., Sandberg, I. 1991. Universal approximation using radial-basis function networks. Neural Compute, 3(2), 264-257.

Park, S., Bae, H., & Lee, J. (2005). End user searching: A Web log analysis of NAVER, a Korean Web search engine. Library & Information Science Research, 27(2), 203-221.

Spink, A., Jansen, B.J., Wolfram, D., and Saracevic, T. (2002). From E-sex to E-commerce: Web search changes. IEEE Computer, 35(3), 133-135.

Swanson, D. R. (1977). Information retrieval as a trial-and-error process. Library Quarterly, 47(2), 128-148.

Appendix A -- MLPN and RBFN Common Java Code

File 1: excelReader.java

File Description: This java file defines a class that can retrieve information from excel database and return the values to defined objects.

```
import java.io.*;

import java.sql.*;

public class excelReader{

        int k=0;

        int rowNum=550; //550 records

        int colNum=10; //9 factors and click through rate

        double[][] data=new double[rowNum][colNum];

public excelReader(){

        Connection connection = null;

        try{

         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

         Connection con = DriverManager.getConnection( "jdbc:odbc:rejection_rate_xls_test" ); // excel file of data

         Statement st = con.createStatement();

         ResultSet rs = st.executeQuery( "Select * from [Sheet1$]" );

         ResultSetMetaData rsmd = rs.getMetaData();

         int numberOfColumns = rsmd.getColumnCount();

         while (rs.next()) {

                 for (int i = 1; i <= numberOfColumns; i++) {

                         data[k][i-1]= rs.getDouble(i);

                 }

                 k=k+1;

         }

         rs.close();

         st.close();

         }

        catch(Exception ex) {

         System.err.print("Exception: ");

         System.err.println(ex.getMessage());

         }

  }
```

```
	public double[][] getData() {
		return data;
	}
}
```

File 2: Net.java

File Description: This java file defines a class of the NNs, which includes the elements like number of layers, input layer, output layer, layer array, momentum factor, learning rate, gain and error.

```
public class Net {
//define layer
	int numLayer=4;
	//define the references of input and output layer
	Layer inputLayer;
	Layer outputLayer;
	Layer Layer[]=new Layer[numLayer]; //define all the layers
	double alpha = 0.9;//momentum factors, which means the proportion of the previous adjusted weights needed to use to adjust the current weight
	double eta = 0.25;//learning rate
	double gain = 1;//gain of sigmoid function
	double error;//total net error
}
```

File 3: Layer.java

File Description: This java file defines a class of the NN's layer, which includes the elements like output, layer error, weight array, saved weight array, adjusted weight array, and the methods like returning the output of the layer, adjusting the errors, returning the saved weights, saving the weights and adjusting the weights.

```
public class Layer {
int units;
double[] output;
double[] error;
double[][] weight;
double[][] weightSave; //saved weights for stopped straining
double[][] dWeight; //last weight deltas for momentum
// construct output, error, weight, adjusted weights memories
public void setOutput() {
	double[] outputTemp=new double[units+1];
	output=outputTemp;
}
public void setError() {
            double[] errorTemp=new double[units+1];
            error=errorTemp;
}

public void setWeight(int i, int j) {
            double[][] setWeightTemp=new double[i][j];
            weight=setWeightTemp;
}

public void setWeightSave(int i, int j) {
            double[][] setWeightSaveTemp=new double[i][j];
            weightSave=setWeightSaveTemp;
}

public void setdWeight(int i, int j) {
            double[][] temp=new double[i][j];
            dWeight=temp;
}
}
```

Appendix B -- MLPN Java Code

File 4: MLP.java

File Description: This java file is the main class of MLPN, which includes training the network, testing the network and evaluating the network by using a constructing technique of MPLN. Firstly, we initialize the network and forward the input to get the output of each neuron in each layer. We use nonlinear transfer functions (e.g. sigmoid nonlinearity function) to learn nonlinear and linear relationships between the input and output vectors. Secondly, we use training data set to train the network until the overall errors tested by the testing data set is low enough (changes are below 20%). We use the DELTA training algorithm to backpropagate the network by apportioning the errors to each unit according to the amount of this error the units are responsible for. Thirdly, we use evaluating data set to evaluate the network through network errors and compare the output with target output. Based on the output and target data, we draw the fitting curve plot.

```
import java.io.FileWriter;

import java.io.PrintWriter;

import java.io.BufferedReader;

import java.io.FileReader;

import java.math.*;


public class MLP {

        double HI=0.9; //the upper limits for the random number and normalized data

        double LO=0.1; //the lower limits for the random number and normalized data

        double bias=1; //bias

        int N=9; //number of input

        int M=1; // number of output

        int numLayer=4;

        int[] units={N,6,4,M};

        int firstRecord = 1;

        int numRecord = 550;

        int trainLWB = 0; //lower bound of training set

        int trainUPB = 299; //upper bound of training set

        int testLWB = 300; //lower bound of testing set

        int testUPB = 499; //upper bound of testing set

        int trainNum = trainUPB-trainLWB+1; //the number of saMLPes in training set

        int testNum = testUPB-testLWB+1; //the number of saMLPes in testing set

        int evalLWB = 500; //lower bound of evaluating set

        int evalUPB = numRecord - 1; //upper bound of evaluating set

        int evalNum = evalUPB-evalLWB+1; //the number of saMLPes in the evaluating set
```

```
//read data
excelReader read = new excelReader();
int rowNum=read.rowNum;
int colNum=read.colNum;
double[][] data=read.getData(); //get data through excel file
double[][] data1=new double[rowNum][colNum];//backup data
double[] dataMean=new double[rowNum]; //used to train the initial network

//define training and testing errors
double trainErrorPredictionMean;
double trainError;
double testErrorPredictionMean;
double testError;

//random drawn from distributions
public int randomEqualINT(int low, int high) {
        return (int)Math.round(Math.random()*(high-low))+low;
}

public double randomEqualDOUBLE(double low, double high){
        return Math.random()*(high-low)+low;
}

//normalize data
public void normalizeData() {
        double min, max;
        for(int k=0;k<=N;k++){
                min=data[0][k];
                max=data[0][k];
                for(int i=1;i<numRecord;i++) {
                        if(data[i][k]<min) {
                                min=data[i][k];
                        }
                        if(data[i][k]>max) {
                                max=data[i][k];
                        }
```

```
            }
            for(int j=0;j<numRecord;j++) {
                data1[j][k]=data[j][k];
                data[j][k]=(data[j][k]-min)/(max-min)*(HI-LO)+LO; //normalize the
data into the range of 0.1 to 0.9
                dataMean[k]=dataMean[k]+data[j][k]/numRecord; //get the mean of
the normalized data
            }
        }
    }
    //initialize the primary elements
    public void initializeApplication(Net Net) {
        double out, err;
        Net.alpha=0.5; //Net momentum factor
        Net.eta=0.05; //Net learning rate
        Net.gain=1; //Net gain
        normalizeData();

        //initialize the training set's errors
        trainErrorPredictionMean=0;
        for (int i=trainLWB; i<=trainUPB; i++) {
            for (int j=0;j<M;j++) {
                out = data[i][N+j];
                err = dataMean[N]-out;
                trainErrorPredictionMean += 0.5*Math.pow(err,2);
            }
        }

        //initialize the testing set's errors
        testErrorPredictionMean=0;
        for (int i=testLWB; i<=testUPB; i++) {
            for (int j=0;j<M;j++) {
                out = data[i][N+j];
                err = dataMean[N]-out;
                testErrorPredictionMean += 0.5*Math.pow(err,2);
            }
        }
    }
```

```
	public void generateNetwork(Net Net) { //allocate the memory to the objects
		int l;
		Net.numLayer = numLayer; //the number of layers
		for(l=0;l<numLayer;l++) {
			Net.Layer[l].units=units[l];
			Net.Layer[l].setOutput();
			Net.Layer[l].output[0]=-bias;		//bias
			Net.Layer[l].setError();
			if(l!=0) {
				Net.Layer[l].setWeight(units[l]+1, units[l-1]+1);
				Net.Layer[l].setWeightSave(units[l]+1, units[l-1]+1);
				Net.Layer[l].setdWeight(units[l]+1, units[l-1]+1);
			}
		}
		Net.inputLayer=Net.Layer[0];
		Net.outputLayer=Net.Layer[numLayer-1];
		Net.alpha=0.9; // Net momentum, the proportion of the last adjusted weights
		Net.eta=0.25; // Net learning rate
		Net.gain=1; //Net gain
	}

	public void randomWeights(Net Net) {
		int l,i,j;
		for (l=1;l<numLayer;l++) { //do not include input layer
			for (i=1;i<=Net.Layer[l].units;i++) {
				for (j=0;j<=Net.Layer[l-1].units;j++) {
					Net.Layer[l].weight[i][j]=randomEqualDOUBLE(-
0.5,0.5);//generate random number between -0.5 and 0.5
				}
			}
		}
	}

	public void setInput(Net Net, double[] input) {
		for (int i=1;i<=Net.inputLayer.units;i++) {
			Net.inputLayer.output[i]=input[i-1];
```

```
    }
}

public void getOutput(Net Net, double[] output) {
    //double[] output=new double[outputLayer.units];
    for (int i=1;i<=Net.outputLayer.units;i++){
        output[i-1]=Net.outputLayer.output[i];
    }
    //return(output);
}

public void saveWeight(Net Net) { //save the current weights to weightSave variable
    int l,i,j;
    for (l=1;l<numLayer;l++) { //do not include input layer
        for (i=1;i<=Net.Layer[l].units;i++) {
            for (j=0;j<=Net.Layer[l-1].units;j++) {
Net.Layer[l].weightSave[i][j]=Net.Layer[l].weight[i][j];
            }
        }
    }
}

public void restoreWeight(Net Net) { //restore the weightSave variable into weights variable
    int l,i,j;
    for (l=1;l<numLayer;l++) { //do not include input layer
        for (i=1;i<=Net.Layer[l].units;i++) {
            for (j=0;j<=Net.Layer[l-1].units;j++) {
Net.Layer[l].weight[i][j]=Net.Layer[l].weightSave[i][j];
            }
        }
    }
}

public void propagateLayer(Net Net, Layer Lower, Layer Upper) { // forward propagate the layer
    int i,j;
    double sum;
    for (i=1;i<=Upper.units;i++) {
            sum =0;
            for (j=0;j<=Lower.units;j++) {
                sum+=Upper.weight[i][j]*Lower.output[j];//sum up the product of
weights and inputs
            }
            Upper.output[i]=1/(1+Math.exp(-Net.gain*sum)); //put the sum into the sigmoid
active function
        }
    }

    public void propagateNet(Net Net) { // forward propagate the Net
        int l;
        for (l=0;l<numLayer-1;l++) {
            propagateLayer(Net,Net.Layer[l],Net.Layer[l+1]);
        }
    }

    public void computeOutputError(Net Net,double[] target) { //compute the output error for the
whole net
        int i;
        double err, out;
        Net.error=0;
        for(i=1;i<=Net.outputLayer.units;i++) {
            out=Net.outputLayer.output[i];
            err=target[i-1]-out;
            Net.outputLayer.error[i]=Net.gain*out*(1-out)*err;
            Net.error+=0.5*Math.pow(err,2);
        }
    }

    //backpropagate the Layers in the Net
    public void backpropagateLayer(Net Net, Layer Upper,Layer Lower) {
        int i,j;
        double out, err;
        for(i=1;i<=Lower.units;i++) {
            out=Lower.output[i];
            err=0;
            for(j=1;j<=Upper.units;j++) {
                err+=Upper.weight[j][i]*Upper.error[j]; //calculate the error for each
hidden layer
            }
            Lower.error[i]=Net.gain*out*(1-out)*err;
        }
    }

    //backpropagate the Net for each hidden layer
    public void backpropagateNet(Net Net) {
        for(int l=numLayer-1;l>1;l--){
            backpropagateLayer(Net,Net.Layer[l],Net.Layer[l-1]);                    }
    }

    public void adjustWeights(Net Net) { //adjust the weights based on the errors
        int l,i,j;
        double out, err, dWeight;
        for(l=1;1<numLayer;l++) {
            for(i=1;i<=Net.Layer[l].units;i++) {
                for(j=0;j<=Net.Layer[l-1].units;j++) {
                    out=Net.Layer[l-1].output[j]; //the same as the input of layer L
                    err=Net.Layer[l].error[i];
                    dWeight=Net.Layer[l].dWeight[i][j];
    Net.Layer[l].weight[i][j]+=Net.eta*err*out+Net.alpha*dWeight; //adjust the weight by
eta*input*error+alpha*last adjusted weights
                    Net.Layer[l].dWeight[i][j]=Net.eta*err*out;
                }
            }
            if(l==numLayer-1) {
                break;
            }
        }
    }

    //simulate the network
    public void simulateNet(Net Net, double[] input,double[] output, double[] target, boolean training)
{
        setInput(Net,input);
        propagateNet(Net);
        getOutput(Net,output);
        computeOutputError(Net,target);
        if(training) {
            backpropagateNet(Net);
            adjustWeights(Net);
        }
    }

    public void trainNet(Net Net,int epochs) { //train Net
        int i,n; //i means start training point
        int j;
        double[] input=new double[N]; //input data
        double[] output=new double[M]; //output data
        double[] target=new double[M]; //target data
        for (n=0;n<epochs*trainNum;n++) { //training epochs
            i=randomEqualINT(trainLWB, trainUPB); //random generate the starting record
            for (j=0;j<N;j++) {//generate input
                input[j]=data[i][j];
            }
            target[0]=data[i][N];
            simulateNet(Net, input, output, target,true);
        }
    }

    public void testNet(Net Net) { //Test Net

        double[] output=new double[M];
        double[] input=new double[N];
        double[] target=new double[M];
        //calculate training set errors
        trainError=0;
        for(int i=trainLWB;i<=trainUPB;i++) {
            for (int j=0;j<N;j++) {//generate training data input
                input[j]=data[i][j];
            }
            target[0]=data[i][N];//since there is only one output needed, for multiple output,
we need loop here
```

```
            simulateNet(Net,input,output,target,false);
            trainError+=Net.error;
        }
        //calculate testing set errors
        testError=0;
        for(int i=testLWB;i<=testUPB;i++) {
            for (int j=0;j<N;j++) {//generate testing data input
                input[j]=data[i][j];
            }
            target[0]=data[i][N]; //since there is only one output needed, for multiple output, we need loop here
            simulateNet(Net,input,output,target,false);
            testError+=Net.error;
        }

        //output the training set and testing set errors
        System.out.println("NMSE on training set is "+trainError/trainErrorPredictionMean+" And NMSE on testing set is "+testError/testErrorPredictionMean);
    }

    public void evaluateNet(Net Net) {//evaluate the Net
        int i,j;
        double[] input=new double[N];
        double[] target=new double[M];
        double[] output=new double[M];
        System.out.println("===Multiple Layer Neural Network Prediction===");
        for(i=evalLWB;i<=evalUPB;i++) {
            for (j=0;j<N;j++) {//generate evaluating open loop data input
                input [j]=data[i][j];
            }
            target[0]=data[i][N];
            simulateNet(Net, input,output,target,false);
            //output the evaluation of the network
            System.out.println((i+firstRecord)+"  "+data[i][N]+"   "+output [0]);
        }
    }
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        boolean stop;
        double minTestError;
        MLP search=new MLP();
        Net Net=new Net();
        //distribute space to layers
        for(int j=0;j<search.numLayer;j++){
            Net.Layer[j]=new Layer();
        }
        //initialize the network
        search.generateNetwork(Net);
        search.randomWeights(Net);
        search.initializeApplication(Net);
        stop = false;
        minTestError =  100000000; //big number
        do {
            search.trainNet(Net,10);
            search.testNet(Net);
            if(search.testError<minTestError) { //if there is an improvement
                System.out.println("...Save weights..."); //save the weight
                minTestError=search.testError;
                search.saveWeight(Net);
            }
            else if(search.testError>1.2*minTestError) { //if there is no improvement and
testerror is bigger than 1.2 times minTestError
                System.out.println("==Stopping training and restoring
weights....");//stop training
                System.out.println("==Final weights to each factors==");
                for(int i=0;i<search.N;i++) {
                    double factorWeight=0;
                    for(int j=1;j<=Net.Layer[1].units;j++) {
    factorWeight+=factorWeight+Net.Layer[1].weightSave[j][i];
                    }
                    factorWeight=factorWeight/Net.Layer[1].units;
                    System.out.println("Factor "+(i+1)+": "+factorWeight);
                }
                stop=true;
                search.restoreWeight(Net); //restore the weights causing the minimum
testing error
```

```
            }
        }while (!stop);
        search.testNet(Net);
        search.evaluateNet(Net);
    }
}
```

Appendix C -- RBFN Java Code

File 5: RBF.java

File Description: This java file is the main class of RBFN, which includes training the network, testing the network and evaluating the network by using a constructing technique of RBFN. Firstly, we use the input data to estimate the kernel positions and kernel widths of neural centers by using an unsupervised clustering algorithm—K-means clustering method. Secondly, we forward the distances between RBF centers and input array to a nonlinear activation function--the radial basis activation function. Thirdly, we forward input to the output layer by using initialized network parameters. Then, we use supervised LMS algorithm to train the network until the overall errors tested by the testing data set is low enough (changes are below 20%). Finally, we use evaluating data set to evaluate network through network errors and compare the output with target. Based on the output and target dataset, we draw the fitting curve plot.

```
import java.io.FileWriter;
import java.io.PrintWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.math.*;
import java.util.ArrayList;
import java.util.Random;

public class RBFN {
        double HI=0.9; //the upper limits for the random number and normalized data
        double LO=0.1; //the lower limits for the random number and normalized data
        double bias=1; //bias
        int centerNum = 5;//center number
        int N=9; //number of input
        int M=1; // number of output
        int numLayer=3; //output layer have three inner layers
        int[] units={centerNum,5,M}; //units for output layers
        int firstRecord = 1; //the first number
        int numRecord = 550;//the last number
        int trainLWB = 0; //the first number for training set
        int trainUPB = 299; //the last number for training set
        int testLWB = 300; //the first number for testing set
        int testUPB = 499; //the last number for testing set
        int trainNum = trainUPB-trainLWB+1; //the number of users included in the training set
        int testNum = testUPB-testLWB+1; //the number of users included in the testing set
        int evalLWB = 500; //the first number of the evaluation set
        int evalUPB = numRecord - 1; //the last number of the evaluation set
        int evalNum = evalUPB-evalLWB+1; //the number of users included in the evaluation set

        //read data
        excelReader read = new excelReader();
        int rowNum=read.rowNum;
        int colNum=read.colNum;
        double[][] data=read.getData(); //get data through excel file
        double[][] data1=new double[rowNum][colNum];//backup data
        double[] dataMean=new double[colNum]; //used to train the initial network
        double[][] centerData=new double[rowNum][centerNum+1]; //used to restore the data coming out
from the center activation functions as well as the final target
        double[] centerDataMean=new double[colNum];
```

```
    //define training and testing errors
    double trainErrorPredictionMean;
    double trainError;
    double testErrorPredictionMean;
    double testError;

    //random drawn from distributions
    public int randomEqualINT(int low, int high) {
        return (int)Math.round(Math.random()*(high-low))+low;
    }

    public double randomEqualDOUBLE(double low, double high){
        return Math.random()*(high-low)+low;
    }

    //normalize data
    public void normalizeData() {
        double min, max;
        for(int k=0;k<=N;k++){
            min=data[0][k];
            max=data[0][k];
            for(int i=1;i<numRecord;i++) {
                if(data[i][k]<min) {
                    min=data[i][k];
                }
                if(data[i][k]>max) {
                    max=data[i][k];
                }
            }
            for(int j=0;j<numRecord;j++) {
                data1[j][k]=data[j][k]; //backup data
                data[j][k]=(data[j][k]-min)/(max-min)*(HI-LO)+LO; //normalize the
data into the range of 0.1 to 0.9
                dataMean[k]=dataMean[k]+data[j][k]/numRecord; //get the mean of
the normalized data
            }
        }
    }

    //normalize center data, the idea is the same as normalizeData function
    public void normalizeCenterData() {
        double min, max;
        for(int k=0;k<centerNum;k++){
            min=centerData[0][k];
            max=centerData[0][k];
            for(int i=1;i<numRecord;i++) {
                if(data[i][k]<min) {
                    min=centerData[i][k];
                }
                if(centerData[i][k]>max) {
                    max=centerData[i][k];
                }
            }
            for(int j=0;j<numRecord;j++) {
                centerData[j][k]=(centerData[j][k]-min)/(max-min)*(HI-LO)+LO;
//normalize the data into the range of 0.1 to 0.9
                centerDataMean[k]=centerDataMean[k]+centerData[j][k]/numRecord;
//get the mean of the normalized data
            }
        }
    }

    //calculate distance between two vectors
    public double distance(double[] input1, double[] input2) {
        double distance;
        double sum=0;
        for(int i=0;i<input1.length;i++) {
            sum+=Math.pow((input1[i]-input2[i]), 2);
        }
        distance=Math.pow(sum, 0.5);
        return distance;
    }

    //generate center
    public ArrayList<double[]> generateCenter(int centerNum){
        //generate the initial center
        ArrayList<double[]> centerList = new ArrayList<double[]>(); //array list to restore the
initial centers
        for(int i=0;i<centerNum;i++) {
            double[] initialCenter=new double[N];
            for(int j=0;j<N;j++) {
                Random r=new Random();
                double rand=r.nextDouble();
                initialCenter[j]=LO+(HI-LO)*rand;
            }
            centerList.add(initialCenter);
        }
        double bigNum=10000000; //big number as the initial value of minimum distance
        int minCenter; // record the index of minimum center
        double eta=0.9; //learning rate for k means clustering
        for(int i=trainLWB;i<trainUPB;i++) {//use the training data set to pick the centers
            //pick the best fit center
            minCenter=0;
            double minDistance=bigNum;
            double[] input=new double[N];
            //get input data
            for (int m=0;m<N;m++) {//generate testing data input
                input[m]=data[i][m];
            }
            //get the closest center
            for (int j=0;j<centerNum;j++) {
                if (distance(input,centerList.get(j))<minDistance) {
                    minDistance=distance(input,centerList.get(j));
                    minCenter=j;//record the closest center
                }
            }
            //updating
            for (int k=0;k<centerNum;k++) {
                if (k==minCenter) { //only chance the closest center w.r.t the input
                    for(int p=0;p<centerList.get(k).length;p++) {
                        centerList.get(k)[p]=(input[p]-
centerList.get(k)[p])*eta+centerList.get(k)[p];
                    }
                }
            }
        }
        //print out clustered centers
        for (int k=0;k<centerNum;k++) {
            System.out.println("center "+k+": ");
            for(int p=0;p<N;p++) {
                System.out.print(centerList.get(k)[p]+" ");
            }
            System.out.println();

        }
        return(centerList);
    }

    //RBF function to screen out the nonlinear activities
    public ArrayList<double[]> activation(ArrayList<double[]> centerList, int LWB, int UPB) {
        //calculate the maximum distance between the centers
        double maxCenterDistance=0; //record the maximum distance
        for (int i=0;i<centerNum;i++) {
            for(int j=0;j<centerNum;j++) {
                if (j!=i) {
                    if (maxCenterDistance<distance(centerList.get(i),
centerList.get(j))) {
                        maxCenterDistance=distance(centerList.get(i),
centerList.get(j));
                    }
                }
            }
        }
        ArrayList<double[]> centerOutputList = new ArrayList<double[]>(); //array list to
restore the clustered centers
        //generate center output array list
        for(int i=LWB;i<UPB;i++) {
            double[] temp=new double[centerNum];
            for(int j=0;j<centerNum;j++) {
                double[] input=new double[N];
                //get input data
                for (int m=0;m<N;m++) {
                    input[m]=data[i][m];
                }
                temp[j]=Math.exp(-(centerNum*Math.pow(distance(input,
centerList.get(j)),2))/Math.pow(maxCenterDistance, 2)); //RBFN function
            }
            centerOutputList.add(temp);
        }
        return (centerOutputList);
    }

    //generate the input putting into the MLP
    public void MLPinput(ArrayList<double[]> input,int centerNum,int LWB, int UPB) {
        for (int i=LWB;i<UPB;i++) {
            for(int j=0;j<centerNum;j++) {
                centerData[i][j]=input.get(i-LWB)[j]; //update the data coming out
from the center RBFN activation function
            }
            centerData[i][centerNum]=data[i][N]; //click through rate
        }
    }

    public void initializeApplication(Net Net) {//initialize the primative elements
        double out, err;
        Net.alpha=0.5; //Net momentum factor
        Net.eta=0.05; //Net learning rate
        Net.gain=1; //Net gain
        normalizeCenterData();
        //initialize the training set's errors
        trainErrorPredictionMean=0;
        for (int i=trainLWB; i<=trainUPB; i++) {
            for (int j=0;j<M;j++) {
                out = centerData[i][centerNum+j];
                err = centerDataMean[centerNum]-out;
                trainErrorPredictionMean += 0.5*Math.pow(err,2);
            }
        }

        //initialize the testing set's errors
        testErrorPredictionMean=0;
        for (int i=testLWB; i<=testUPB; i++) {
            for (int j=0;j<M;j++) {
                out = centerData[i][centerNum+j];
                err = centerDataMean[centerNum]-out;
                testErrorPredictionMean += 0.5*Math.pow(err,2);
            }
        }
    }

    //generate network
    public void generateNetwork(Net Net) { //allocate the memory to the objects
        int l;
        Net.numLayer = numLayer; //the number of layers
        for(l=0;l<numLayer;l++) {
            Net.Layer[l].units=units[l];
            Net.Layer[l].setOutput();
            Net.Layer[l].output[0]=-bias;       //bias*(-1)
            Net.Layer[l].setError();
            if(l!=0) {
                Net.Layer[l].setWeight(units[l]+1, units[l-1]+1);
                Net.Layer[l].setWeightSave(units[l]+1, units[l-1]+1);
                Net.Layer[l].setdWeight(units[l]+1, units[l-1]+1);
            }

        }
        Net.inputLayer=Net.Layer[0];
        Net.outputLayer=Net.Layer[numLayer-1];
        Net.alpha=0.9; // Net momentum, the proportion of the last adjusted weights
        Net.eta=0.25; // Net learning rate
        Net.gain=1; //Net gain
    }
    //generate random weights
    public void randomWeights(Net Net) {
        int l,i,j;
        for (l=1;l<numLayer;l++) { //do not include input layer
            for (i=1;i<=Net.Layer[l].units;i++) {
                for (j=0;j<=Net.Layer[l-1].units;j++) {
                    Net.Layer[l].weight[i][j]=randomEqualDOUBLE(-
0.5,0.5);//generate random number between -0.5 and 0.5
                }
            }
        }
    }

    //set input for each units of input layer
    public void setInput(Net Net, double[] input) {
        for (int i=1;i<=Net.inputLayer.units;i++) {
            Net.inputLayer.output[i]=input[i-1];
        }
    }

    //set output for each units of output layer
    public void getOutput(Net Net, double[] output) {
        for (int i=1;i<=Net.outputLayer.units;i++){
            output[i-1]=Net.outputLayer.output[i];
        }
    }

    //save the current weights to weightSave variable
    public void saveWeight(Net Net) {
        int l,i,j;
        for (l=1;l<numLayer;l++) { //do not include input layer
            for (i=1;i<=Net.Layer[l].units;i++) {
                for (j=0;j<=Net.Layer[l-1].units;j++) {
                    Net.Layer[l].weightSave[i][j]=Net.Layer[l].weight[i][j];
                }
            }
        }
    }

    //restore the weightSave variable into weights variable
    public void restoreWeight(Net Net) {
        int l,i,j;
        for (l=1;l<numLayer;l++) { //do not include input layer
            for (i=1;i<=Net.Layer[l].units;i++) {
                for (j=0;j<=Net.Layer[l-1].units;j++) {
                    Net.Layer[l].weight[i][j]=Net.Layer[l].weightSave[i][j];
                }
            }
        }
    }

    // forward propagate the layer
    public void propagateLayer(Net Net, Layer Lower, Layer Upper) {
        int i,j;
        double sum;
        for (i=1;i<=Upper.units;i++) {
            sum =0;
            for (j=0;j<=Lower.units;j++) {
                sum+=Upper.weight[i][j]*Lower.output[j];//sum up the product of
weights and inputs
            }
            Upper.output[i]=Net.gain*sum/Upper.units; //for RBFN linear output layer
        }
    }

    // forward propagate the Net
    public void propagateNet(Net Net) {
        int l;
        for (l=0;l<numLayer-1;l++) {
            propagateLayer(Net,Net.Layer[l],Net.Layer[l+1]);
        }
    }

    //compute the output error for the whole net
    public void computeOutputError(Net Net,double[] target) {
        int i;
        double err, out;
        Net.error=0;
        for(i=1;i<=Net.outputLayer.units;i++) {
            out=Net.outputLayer.output[i];
            err=target[i-1]-out;
            Net.outputLayer.error[i]=Net.gain*out*(1-out)*err;
            Net.error+=0.5*Math.pow(err,2);
        }
    }

    //backpropagate the Layers in the Net
    public void backpropagateLayer(Net Net, Layer Upper,Layer Lower) {
        int i,j;
        double out, err;
        for(i=1;i<=Lower.units;i++) {
            out=Lower.output[i];
            err=0;
            for(j=1;j<=Upper.units;j++) {
                err+=Upper.weight[j][i]*Upper.error[j]; //calculate the error for each
hidden layer
            }
            Lower.error[i]=Net.gain*out*(1-out)*err;
        }
    }

    //backpropagate the whole Net
    public void backpropagateNet(Net Net) {
        for(int l=numLayer-1;l>1;l--){
            backpropagateLayer(Net,Net.Layer[l],Net.Layer[l-1]); //backpropagate the Net
for each hidden layer
        }
    }

    //adjust the weights based on the errors
    public void adjustWeights(Net Net) {
        int l,i,j;
        double out, err, dWeight;
        for(l=1;1<numLayer;l++) {
            for(i=1;i<=Net.Layer[l].units;i++) {
                for(j=0;j<=Net.Layer[l-1].units;j++) {
                    out=Net.Layer[l-1].output[j]; //the same as the input of layer L
                    err=Net.Layer[l].error[i];
                    dWeight=Net.Layer[l].dWeight[i][j];
                Net.Layer[l].weight[i][j]+=Net.eta*err*out+Net.alpha*dWeight;
//adjust the weight by eta*input*error+alpha*last adjusted weights
                    Net.Layer[l].dWeight[i][j]=Net.eta*err*out;
                }
        }
            if(l==numLayer-1) { //to avoid computation exceptions
                break;
            }
        }
    }

    //simulate the network
    public void simulateNet(Net Net, double[] input,double[] output, double[] target, boolean training)
{
        setInput(Net,input);
        propagateNet(Net);
        getOutput(Net,output);
        computeOutputError(Net,target);
        if(training) {
            backpropagateNet(Net);
            adjustWeights(Net);
        }
    }

    public void trainNet(Net Net,int epochs) { //train Net
        int i,n; //i means start training point
        int j;
        double[] input=new double[centerNum]; //input data
        double[] output=new double[M]; //output data
        double[] target=new double[M]; //target data

        for (n=0;n<epochs*trainNum;n++) { //training epochs
            i=randomEqualINT(trainLWB, trainUPB); //random generate the starting record
            for (j=0;j<centerNum;j++) {//generate input
                input[j]=centerData[i][j];
            }
            target[0]=centerData[i][centerNum];
            simulateNet(Net, input, output, target,true);
        }
    }

    public void testNet(Net Net) { //Test Net
        double[] output=new double[M];
        double[] input=new double[centerNum];
        double[] target=new double[M];

        //calculate training set errors
        trainError=0;
        for(int i=trainLWB;i<=trainUPB;i++) {
            for (int j=0;j<centerNum;j++) {//generate training data input
                input[j]=centerData[i][j];
            }
            target[0]=centerData[i][centerNum];//since there is only one output needed, for
multiple output, we need loop here
            simulateNet(Net,input,output,target,false);
            trainError+=Net.error;
        }


        //calculate testing set errors
        testError=0;
        for(int i=testLWB;i<=testUPB;i++) {
            for (int j=0;j<centerNum;j++) {//generate testing data input
                input[j]=centerData[i][j];
            }
            target[0]=centerData[i][centerNum]; //since there is only one output needed, for
multiple output, we need loop here
            simulateNet(Net,input,output,target,false);
            testError+=Net.error;
        }

        //output the training set and testing set errors
        System.out.println("NMSE on training set is "+trainError/trainErrorPredictionMean+"
And NMSE on testing set is "+testError/testErrorPredictionMean);
    }

    public void evaluateNet(Net Net) {//evaluate the Net
        int i,j;
        double[] input=new double[centerNum];
        double[] target=new double[M];
        double[] output=new double[M];
        System.out.println("====RBFN Rredicton=====================");
        for(i=evalLWB;i<=evalUPB;i++) {
            for (j=0;j<centerNum;j++) {//generate evaluating open loop data input
                input[j]=centerData[i][j];
            }
            target[0]=data[i][N];
            simulateNet(Net,input,output,target,false);
            System.out.println((i+firstRecord)+"  "+data1[i][N]+"   "+output[0]);
        }
    }


    public static void main(String[] args) {
        // TODO Auto-generated method stub
        boolean stop;
        double minTestError;
        RBFN search=new RBFN();
        Net Net=new Net();
        //distribute space to layers
        for(int j=0;j<search.numLayer;j++){
            Net.Layer[j]=new Layer();
        }
        search.normalizeData();//normalize the original data
        //generate the center and the output coming from the center RBFN activation functions
        ArrayList<double[]> centerList = new ArrayList<double[]>();
        centerList=search.generateCenter(search.centerNum);
        ArrayList<double[]> RBFoutput = new ArrayList<double[]>();
        RBFoutput=search.activation(centerList,search.trainLWB,search.trainUPB);
        search.MLPinput(RBFoutput,search.centerNum,search.trainLWB,search.trainUPB);
        RBFoutput=search.activation(centerList,search.testLWB,search.testUPB);
        search.MLPinput(RBFoutput,search.centerNum,search.testLWB,search.testUPB);
        RBFoutput=search.activation(centerList,search.evalLWB,search.evalUPB);
        search.MLPinput(RBFoutput,search.centerNum,search.evalLWB,search.evalUPB);
        //initialize the network
        search.generateNetwork(Net);
        search.randomWeights(Net);
        search.initializeApplication(Net);
        stop = false;
        minTestError =  100000000; //big number
        do {
            search.trainNet(Net,10);
            search.testNet(Net);
            if(search.testError<minTestError) { //if there is an improvement
                System.out.println("...Save weights..."); //save the weight
                minTestError=search.testError;
                search.saveWeight(Net);
            }
            else if(search.testError>1.2*minTestError) { //if there is no improvement and
test error is bigger than 1.2 times minTestError
                System.out.println("==Stopping training and restoring
weights....");//stop training
                System.out.println("==Final weights to each factors==");
                for(int i=0;i<search.centerNum;i++) {
                    double factorWeight=0;
                    for(int j=1;j<=Net.Layer[1].units;j++) {
    factorWeight+=factorWeight+Net.Layer[1].weightSave[j][i];
                    }
                    factorWeight=factorWeight/Net.Layer[1].units;
                    System.out.println("Factor "+i+": "+factorWeight);
                }
                stop=true;
                search.restoreWeight(Net); //restore the weights causing the minimum
testing error
            }
        }while (!stop);
        search.testNet(Net);
        search.evaluateNet(Net);
    }

}
```