PERSONALIZED RECOMMENDER SYSTEM

ON WHOM TO FOLLOW IN TWITTER

by

Masudul Islam

B.Sc. in Computer Science, University of Toronto, Canada, 2006

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2014

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

MASUDUL ISLAM

PERSONALIZED RECOMMENDER SYSTEM

ON WHOM TO FOLLOW IN TWITTER

Masudul Islam

Master of Science, Computer Science, Ryerson University, 2014

ABSTRACT

Recommender systems have been widely used in social networking sites. In this thesis, we propose a novel approach to recommend new followees to Twitter users by learning their historic friends-adding patterns. Based on a user's past social graph and her interactions with other connected users, scores based on some of the commonly used recommendation strategies are calculated and passed into the learning machine along with the recently added list of followees of the user. Learning to rank algorithm then identifies the best combination of recommendation strategies the user adopted to add new followees in the past. Although users may not adopt any recommendation strategies explicitly, they may subconsciously or implicitly use some. If the actually added followees match with the ones suggested by the recommendation strategy, we consider users are implicitly using that strategy. The experiment using the real data collected from Twitter proves the effectiveness of the proposed approach.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor Dr. Cherie Ding for her continuous guidance throughout my research. She has been very helpful during the course of my study by showing me the proper direction, providing feedback, and helping me overcome all the challenges and difficulties in my work. I found her suggestion, knowledge and research skills a significant contribution in the completion of this thesis.

I would like to thank all the administrative and technical members of the Department of Computer Science for their cooperation, help and for giving me access to additional resources required for my research.

I would also like to thank Dr. Alex Ferworn, Dr. Alireza Sadeghian and Dr. Eric Harley for taking the time to review my thesis and for providing valuable feedback which enabled in the improvement of the thesis.

Finally, I would like to express my deepest appreciation to my family, friends and relatives for all the motivation and support during the course of my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1
INTRODUCTION

1.1 Background and the Problem Statement

1.1.1 Background

Any form of social experience via the interactions between different people can be defined as social networking. Online social networking has been very popular over the last few years due to the ease of using it and the benefits people are getting from it. However, the concept of social networking is not new. People have been using different forms of social networking for many years and have benefitted from it. The present tools we use for online social networking such as Facebook, Twitter, LinkedIn, have been built on top of the concept of the traditional social network [1].

People get involved in social networking for many reasons, such as: the need to keep in touch with friends and family, the curiosity to know about how others are doing, the urge to know about more information (information seeker), for information giving [2] and so on. Social networking is not only used by people for recreation during lunch breaks and evenings, but it is also used for business. It has proved to be a great tool for marketing and political campaigns. Due to the availability of various internet devices such as desktops, laptops, mobile phones and tablets, participating in social networking has become easier in recent years. As a result, social networks became very popular [1].

In social networking, users acquire new information as well as distribute new information by means of the social links, in other words, by means of the friends. The different reasons people use social network play a great role in adding new friends. Based on the interest of a user, she adds new friends and follows their contribution of information to the social network. As the user strives to expand her network of information, she needs to discover new users. It is often a time consuming task to search for new friends whom the user wants to follow. This is when a recommender system proves to be useful. Almost every social networking site offers some kind of recommender system that recommends new friends to the user. As the network of the user grows and as the interaction of the user increases, recommender systems find more information about the user to add to the model used to discover and recommend new friends to the user. Recommender systems play a great role in improving the overall experience of a user participating in social networking. Recommender system increases social interaction, expands user's social network and saves valuable time of the user.

Twitter is one of the most popular social networking sites in recent times. It is classified as a micro-blogging site. It was launched in October 2006 and became popular after it won the South West (SXSW) conference Web Awards in March 2007 [2]. It is a micro-blogging platform that provides a light-weight easy form of information sharing system where users can broadcast their messages to their followers. The message consists of a maximum of 140 characters and is visible to the user and her followers [3]. Any person can follow another person and unlike Facebook, the users do not have to know each other personally. Use of Twitter has gained popularity recently and it is used for different purposes such as political campaign, education, advertising, marketing, customer service, emergency news, status update and public relation [3].

Compared to blogging, this is a faster method of communication. As a result, the frequency of posting messages is high. Some of the user intentions are daily chatter, conversations, information sharing and news reporting [2]. Friendship links in Twitter is different compared to other social networking site like Facebook. Twitter is a directed social network. In Facebook, the friendship link is bidirectional whereas, in Twitter, the link is unidirectional [4]. A person may follow another person, but the second person may not follow back the first person. In this case we call the first person a follower of the second person, and the second person a followee of the first person. In Twitter, instead of friends, a user has followers and followees. Followers are users who follow this user and followee are the users who this user follows. In some of the Twitter based research papers, the authors referred to followees as friends, while in others authors described followees as friends-to-follow. In this thesis, we will use the term "followee" consistently. Some users play the role of information source while others play that of information seekers [2]. Information source users have more followers than followees; whereas information seekers follow other users more. As a result, information sources post tweets more often than information seekers.

1.1.2 Problem Statement

The satisfaction of a social network user depends on the people she is linked to. While a user finds valuable information from the followee she is following, others following this user also benefit from this user's information. While adding a right followee can be very productive, adding a wrong followee can lead to getting duplicate, irrelevant and sometimes wrong information. Users need to be careful of whom they are following for information. Searching for the right people for the right purpose can be very time consuming.

With the growth of the internet, users can often get overwhelmed with information. Users often rely on recommender systems to discover new followee. There have been many recommendation systems used in the past and the existing recommender systems are continuously being improved to generate the list of new prospective followees whom the user will find interesting. Recent improvements of recommender systems [5] [6] utilize more information than before and find better recommendations using both network links and user content. Researchers have been working continuously to find even better ways to recommend.

There are many strategies being used to recommend new friends to follow. Some strategies use existing friendship links, some use content and collaborative filtering approach, some use social influence, some use topic similarity and so on. There are some recommender systems [3] [7] that use more than one of the above described strategies. In spite of having so many recommendation strategies and systems in use today, there still seems to be a gap between what the users are looking for and what the recommender systems are offering.

Often recommender systems utilize more than one strategy and generate a long list of recommendations. These recommended followees could be ranked, sorted and presented to the user according to the generated score. However, the score generated by the recommender system may not match with the importance given by the user for each of the recommendations. Often times, it is not important how the recommender system is ranking a particular recommendation but how interesting the user thinks it is. A particular recommended followee may be near the bottom of the recommended list, but it can be something that will interest the user.

Some recommendation strategies may use the existing friendship links of the user and recommend followees based on common connections; while other strategies can be collaborative and produce result based on shared interests. Depending on the classification of users, some may

prefer to add common followees, while some may prefer to add a followee who shares similar interests. The particular strategy or the group of strategies a recommender system is using may work for some users but it may not work for others.

The reason users still browse through the long list of recommendations and accept only a few of them is that the recommender system cannot properly predict the user's preferences and needs using the current one-fit-all algorithm. Since there is no single recommendation strategy that works the best for all users, it is still a challenge for the recommender systems to identify the strategy that will work the best for a particular user. If the system can identify the strategy or a combination of strategies that will work for a particular user, then the recommendations will have better chance to be liked by the user. In this thesis, we propose an approach to identity the strategy or combination of strategies that will work the best for a particular user. Our proposed approach makes use of the history data to identify the strategies the user adopted in the past to add new followees. Not necessarily will the user follow the same strategies all the time, but chances are more that the user will follow the same strategies that she has followed them in the past. Also by constantly observing user's pattern, the newly adopted strategies can also be identified once an enough amount of history data is accumulated.

1.2 Objectives

In this thesis, we have two main objectives. Firstly, we want to study users, their previous actions and their followee adding patterns. Individual users have their own patterns of adding new followees. By observing their past patterns, we want to identify if they have employed a single strategy or a combination of strategies in order to add a new followee. Secondly, we want to illustrate that we can achieve better performance by using machine learning technique to learn

the ranking model from multiple recommendation strategies than the case when using any single strategy or just linearly combining all the strategies. Using one or many recommendation strategies, we can get a list of recommendations; however, the effectiveness of the ranking order of the recommended list can only be verified if the ranked recommended list is compared with the list of actually added followees by the user. Instead of using only one recommended strategy, or using a linear combination of all the strategies, we want to use scores from all the recommended strategies and build a model using learning to rank algorithms, a type of machine learning algorithm specially designed for ranking problems.

1.3 The Proposed Methodology

There are many types of social networking sites available today. People have the choice of selecting which one to use. Not only the differences in brand names, but the differences in their functions make the social networking sites unique. In our research we consider Twitter as our social network platform and our experiment uses real data from Twitter. However, the concepts and methodologies we discuss are not limited to Twitter and can apply to any social networking site using recommender systems.

From helping in searching for a long lost friend to helping in discovering the influential expert on a particular topic, recommender systems play great roles in discovering valuable resources to the users of social networking. Our aim in this thesis is to improve the effectiveness of the recommender systems for whom-to-follow in Twitter.

Recommender systems analyze users' classification, network, links, interactions, usage, content, location, and many other features and predict the possibility of the user following another user in future. For some of the strategies, the system may just need to analyze the

existing friendship links of the user and extract common links, while in others, the system may need to analyze hundreds and thousands of user's tweets and content and compare them to that of millions of users for similarity. There are some strategies that analyze random walk pattern e.g. PageRank, SALSA, and deduce the influential leaders or topic experts. Depending on the type of analysis and strategy, the data requirement can be simple; however, it can be complex as well. We will be using Twitter API to collect all the input data for our experiment.

In this thesis, we do not propose to re-invent any new strategy, nor do we propose to improve any particular strategy. We consider some known common strategies, and use them to analyze user's data and come up with a model that will help identify which strategies were used by the user in the past. Instead of presenting an idea for building a new strategy, we present the idea of identifying which strategy or strategies were useful for the user in the past.

We select a handful of strategies used in recommender systems. Using each of our selected strategies, we build a recommendation list consisting of recommended users and their corresponding scores. The items in the recommended list are users who this user does not currently follow but possibly follows in the future. We use historic data as input for the strategies. Once we have the lists of recommended users and their scores, we combine all the lists into one list. In this combined list we have the list of all the recommended users from all the strategies and their scores from all different strategies. Currently we are using 7 different strategies. However, this list can be expanded in the future if necessary.

Our next step is to use the current data to identify how many of our recommended items from our final list are added as friends-to-follow by the user. We modify our previously compiled final list with a flag that represents if the recommended item in the list was later added by the user as a followee. Now that we have a list of recommended users with scores and the flag

to represent the successful followee, we use this information to build a model using popular learning to rank algorithms in our learning machine. Among the learning to rank algorithms that we consider are MART, Coordinate Ascent and Random Forests. Instead of using one particular strategy or all strategies for a user, we use the scores from all the strategies as input to the ranking algorithm and let the learning machine build the model to rank the recommendations. In order to increase the accuracy of our model and increase the amount of training data, we use multiple sets of historic data. We select a list of seed users from a list of Ryerson followers and collect their data using Twitter API for 3 months for our experiment.

1.4 Contributions

The main contributions of our research are as follows:

Firstly, we propose a recommender system that uses learning to rank algorithm to build a personalized candidate ranking model using target user's historic data. To the best of our knowledge, it is one of the earliest efforts to use machine learning algorithms to build a personalized Twitter recommender system.

Secondly, we propose a novel strategy for the recommender system. It calculates the PageRank of the social network users using Retweet links, whereas most of the other Twitter related research work applies PageRank on the following links.

Thirdly, we design and implement an application that consists of a crawler, data processor and a recommender system. The recommender system uses our proposed approach and this system is used in our experiment.

Lastly, we collect real Twitter data for a period of 3 months. The collected data consists of the snapshots of the social network graph of the same users at different timestamps. We use the data for our research and it is available to our research team for any future work.

1.5 Thesis Outline

The remainder of the thesis is organized as follows:

Chapter 2 reviews the work previously done by researchers in this field of interest. This constitutes the literature review of some of the work previously done in social networking and recommender systems in general. A number of different recommendation strategies are reviewed.

Chapter 3 explains the system architecture, the collection of historic data, analysis of the data and algorithms of the recommendation strategies used in our thesis in details.

Chapter 4 explains the experiment to evaluate our proposed model by describing the details of data collection, data preparation, score calculation, machine learning and result analysis.

Finally, Chapter 5 concludes our thesis with a summary of our experimental results and our future research direction.

CHAPTER 2
RELATED WORKS

In this chapter, we discuss the concepts and core logic of recommendation algorithms for social networking that have been used in our thesis and then review some of the recent work done by the researchers in the area of social networking and recommender systems. We review the literature of the models proposed for different recommender strategies including the ones that are currently used for Twitter's "Who-To-Follow" recommender system.

2.1 Background

In this subsection, we discuss some of the concepts and algorithms used in our thesis.

2.1.1 PageRank

PageRank was first introduced by Brin and Page [8] to be used for Google web page search. PageRank is a link analysis technique that is used to examine the structure of a network graph and assign scores to the nodes in a graph. Nodes in the graph represent the web pages. PageRank is used to measure the importance of the nodes in a graph in terms of influence and/or connectivity. Scores range from 0 to 1 and are calculated by random walk based on the topological structure of the graph. The random walk starts at a node and at every step, it goes to any of the connected nodes depending on the probability of visiting a node. Since the probabilities of visiting nodes are different, some nodes will be visited more than the others. Probability of visiting a node depends on the incoming links from other nodes.

If web page A has incoming links from pages $T_1,...T_N$, then the PageRank of A is described as:

$$PR(A) = \frac{(1-d)}{N} + d \sum_{i=1}^{N} \frac{PR(T_i)}{C(T_i)} \quad (2.1)$$

In (2.1), d is defined as the damping factor, which is usually set between 0 and 1, but mostly 0.85; $C(T_i)$ is defined as the number of outgoing links from page T_i and N is the total number of web pages. Since PageRank forms a probability distribution, the sum of PageRanks of all pages equals 1.

If a node has no incoming links from any other node, it doesn't mean that it will not be visited at all. Consider a sample web site, where the web pages are connected to each other by internal links. There can be a page that is not linked from any page but the page may still be visited by a user if the user types the URL of the page in the location bar of the browser and goes to the page directly. This operation is known as teleport [9] or random jumps [6]. The damping factor mentioned by the authors of [8] allows random jumps to another node in the graph.

PageRank can be calculated using Markov chain in combination of The Power Method. It is a process that involves creating an $N*N$ transition probability matrix P where N represents the number of nodes in the graph [9]. The procedure to calculate the value of each element in the matrix can be formulated as:

$$\mathrm{P}(i,j) = \frac{1-d}{N} + \frac{d * linkCounts(i,j)}{outDegree(i)} \quad (2.2)$$

where i represents the row number of the transition matrix, j represents and the column number of the transition matrix, $linkCount(i,j)$ represents the number of links going from node i to node j,

outDegree(i) represents the total number of links from node *i* to other nodes, *d* represents the damping factor and *N* represents the total number of nodes in the graph.

Once the transition probability matrix *P* is calculated, the PageRank score is calculated using an iterative algorithm. In the first iteration, a *1*N* identity matrix is multiplied with *P*. The resulting matrix in this first iteration is used in the second iteration to be multiplied with *P* again. The *1*N* identity matrix in the first iteration represents that starting point of the random walk, which could be the first node in the matrix. The elements of the resulting *1*N* matrix represent the PageRank score of the nodes in the graph.

$$PR_i = P * PR_{i-1} \tag{2.3}$$

where PR_i represents the PageRank score after i^{th} iteration, *P* represents the transition probability matrix and PR_{i-1} represents the PageRank score after $(i-1)^{th}$ iteration. PageRank values are supposed to converge after a certain number of iterations. In implementations, either the program checks for convergence or the number of iterations is specified [9].

2.1.2 Vector Space Model

In this research, Vector Space Model is used to calculate the user similarity based on the tweets they posted. Vector Space Model [10], often called term vector model, is a model used for representing a document as a vector. A document can be considered as a sequence of words, where each word has been mentioned a certain number of times. The words in the document are often referred to as terms. The dimension of the vector in the model will correspond to the number of unique words in the document. For each unique word in the document, there will be a corresponding dimension with a non-zero value. The magnitude of a dimension corresponds to

the weight of the term. The weight of a term is assigned based on the frequency of occurrence of the term. The weight represents the significance of the term in the document.

Vector Space Model is used to calculate similarity between two documents. The two documents are represented by their vectors and similarity is compared by measuring the angle between the vectors, a term known as Cosine Similarity [10].

$$\text{Cosine Similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{||A||||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (2.4)$$

In (2.4), A and B are the term vectors and $||A||$ and $||B||$ are magnitudes of vectors A and B respectively. It is the dot product of two vectors over the product of the magnitude of the vectors. The value of cosine similarity ranges from 0 to 1. A score of 0 represents that the documents don't match and a score of 1 represents that the documents are a perfect match.

2.1.3 Recommendation algorithm in general

Recommendation algorithms can be categorized by the type of data used for generating recommendations. The three main categories of recommender systems are content-based, collaborative filtering based and hybrid system that combines both approaches.

In content-based recommender system, items that are similar to the items liked by the user in the past are recommended. Characteristics and different attributes of the items previously liked by the user are extracted by analyzing content from the user's profile and interactions. The characteristics and values of these attributes are then used to search for similar items which are recommended to the user [11].

Collaborative Filtering based recommender system uses opinions of other users for recommendation. It is one of the most popular approaches and one of the oldest techniques. It does not rely on the content of the item but on what others thought of it. People usually get motivated to do something such as watch a movie or try a different restaurant if they hear good opinions about it from their friends, coworkers, family members or neighbors. Over time, we develop our circle of trust and identify members of the society who have the taste and opinion similar to ours. These members may have opinions similar to ours about the items we both saw or things we both experienced such that we involuntarily trust their opinions. In future, if they have a positive opinion of something that we have not tried, then the chances are we will like it too. On the other hand, if they have a negative opinion, then the chances are we may not like it and we tend to avoid such a thing. Instead of utilizing opinions of just any user, collaborative filtering approach focuses on opinions of similar users [12].

Since Collaborative Filtering depends on the existing opinions, it becomes difficult to rate a new item that no one has rated before. This phenomenon is known as the "cold start" problem. To avoid such problem, a hybrid approach is often used for recommender systems. Both content-based filtering and collaborative-based filtering is used in the hybrid approach. Hybrid approaches try to take advantages of both the content-based and the collaborative approaches [12].

2.1.4 Learning to Rank

2.1.4.1 Overview

Learning to rank [14] is the process of using machine learning techniques to build a ranking model. In this supervised or semi-supervised machine learning process, three sets of data are used to build the model. The sets are: training, validation and test. Training set is used to build the model, validation set is used to refine the model and test set is used to measure the performance and accuracy of the model. The learning to ranking algorithm was initially proposed to solve the ranking problem for information retrieval. In these systems, data consists of a list of documents, scores for a set of features for each of the documents and a relevancy factor for each of the documents. While the scores of the features are characteristics of the documents, the relevance factor shows the importance of the document for the particular query. Thus the same document may have different relevance factors for different queries. The final ranking model is used to rank a list by assigning a relevance score to each of the items in the list usually on their importance with respect to the query. Learning to rank is not only used for information retrieval systems, but also used in recommender systems to further rank the items in the recommended list, and many other domains. When used in information retrieval, the basic unit to rank is a document, while in our system the basic unit is a user who is a potential followee to the target user.

There are three major categories of learning to rank algorithms. They are: point-wise, pair-wise and list-wise [14].

Point-wise algorithms take feature vectors of the documents to rank as input and predict the relevancy scores. They learn the score directly using decision tree models and linear regression.

Algorithms can be regression-based, ordinal regression-based or classification-based. Regression-based algorithms are good in predicting continuous values. Classification-based algorithms are good in predicting discrete values. Association of the document with the query is often ignored in Point-wise. The algorithms do not consider the inter-relationship between documents. While relevancy of some documents may be very dependent on the query, it may not be dependent for some documents. The point-wise algorithms focus on generating the relevancy score rather than focus on the actual position of a document in the ranked list. Thus, some unimportant feature-score may be given more preference than required. Examples of point-wise algorithm include MART and Random Forests [14].

Pair-wise algorithms focus on the relative importance of the documents in a pair rather than the degree of relevance. This type of algorithm takes in a pair of documents that are represented by their feature vectors and predicts the relevancy between the documents in the pair. They work by modeling the classifiers to predict which one of the two documents is more relevant. They consider the relevancy order of the two documents; however, the relevancy order of these documents in the whole rank list can be different. Examples of pair-wise algorithms include RankBoost and RankNet [14].

List-wise algorithms take in the entire list of documents of a particular query and generate ranked list of documents. The content of the output is same as that of the input but in different order. Examples of list-wise algorithms include AdaRank and ListNet.

2.1.4.2 Ranking Algorithms

There are several learning to rank algorithms that are used by researchers today. Each differs in their internal algorithms and their ways of training data for building the ranking model. Below we will review a few that are used in our experiment.

MART (Multiple Additive Regression Trees) [15] is a non-linear point-wise method. It is the commercial implementation of Gradient Boosting Regression tree. It iteratively builds a model in the form of decision tree by optimizing the differentiable loss function in every step. The aim of this ranking algorithm is to estimate a function that satisfies the input and output values of the training data set. The loss function measures the deviation of the output computed by the estimated function from the expected output. This algorithm has been proved to be efficient for web search ranking data. One of the drawbacks is the longer training time compared to the list-wise methods [16].

RankNet [17] is a linear pair-wise learning to rank algorithm proposed by Burges et al. in 2005. It uses neural network to model the ranking function and gradient descent to optimize it. It features a probabilistic cost function. RankNet performs well with a two layered neural network and is efficient for large amount of data.

Freund et al. [18] proposed RankBoost in 1998. It has been built on the basis of Support Vector Machines (SVM). It uses boosting and ranks by minimizing classification errors. It is a pair-wise ranking algorithm and uses a linear combination of the ranking features and attempts to find correlations between weak rankings in order to construct the scoring function.

AdaRank was proposed by Xu and Li [19] in 2007. It uses the list-wise approach and is very similar to RankBoost. However, in contrast to RankBoost, it aims at minimizing the loss

function based on direct optimization of performance measures. In each iteration, it repeatedly re-weighs the training data. It has been evolved from AdaBoost which works for binary classification prediction. AdaRank is used for informational retrieval and document ranking. It is more efficient than RankBoost.

Coordinate Ascent is a list-wise learning to rank algorithm. It was proposed by Metzler and Croft [20] in 2007. It maximizes an evaluation metric such as mean average precision. The scoring function is modelled by linear combination of the feature vectors. The algorithm iterates through the training data trying the different combinations of the feature vectors until convergence takes place. Coordinate Ascent is good for a wide range of data types.

In 2007, Cao et al. [21] propose ListNet. It uses the list-wise approach that ranks the entire list of input. A probabilistic loss function is used as an evaluation measure. It uses neural network for modelling and gradient descent for optimizing. ListNet is very similar to RankNet except that RankNet uses the pair-wise approach. Time complexity of ListNet is $O(mn)$ compared to that of RankNet which is $O(mn^2)$, where m is the number of training queries and n is the maximum number of documents. As a result, ListNet ranks faster than RankNet [14].

Random Forests [22] is a non-linear point-wise learning to rank algorithm. It is similar to Gradient Boosted Regression Tree in that it uses the tree averaging; however, it is less expensive than MART. It takes random selection of features from the training set and uses decision tree classifiers. Parallel algorithm is employed here as the formation of a tree is not dependent on the trees earlier constructed. Random Forests converges as the number of trees increases. In addition, it does not overfit with the increased number of trees. Overfitting happens when the algorithm makes the ranking function too customized for the training data such that the function produces

very little error for the training set but produces larger error for the test data. This algorithm was proposed as an improvement to the existing ranking algorithm AdaBoost. In comparison to AdaBoost, this produces faster result. In addition, it yields better result than AdaBoost in the presence of noise in the training data.

2.2 Related Work

In this section, we review the related work done by researchers in this field of interest. We start with discussing some of the recommender systems proposed for social network in general followed by the ones that are specifically proposed for Twitter.

2.2.1 Recommender systems in social network

Chen et al. [5] propose a recommender system for IBM's social networking site, Beehive system. They implemented the recommender system using four different algorithms: Content matching, Content-plus-Link (CplusL), Friend-of-Friend (FoF), and SONAR. In content matching, the user's content is matched with other users to generate recommendations. Content-plus-Link is an enhanced version of content matching where content-matching is applied to a network of up to 4 members where the first member is connected to the 4th member via the other members. Friend-of-Friend algorithm is a very popular algorithm where if *A* and *B* are friends, then *A* is recommended as a candidate to *B*'s friends. The last algorithm, SONAR, used information from other resources such as organizational charts, publication databases, patent database, friending systems, people tagging system, project wiki and blogging systems. The authors find that Content-matching and Content-plus-Link are useful to recommend mostly unknown users whereas friend-of-friend and SONAR are useful to recommend mostly known

users. It was found that users are more interested to get recommendations of the known users in comparison to the unknown users; however, in some cases users expressed interest in finding unknown related users. The authors suggest use of social-relations-based algorithms for a new social network and use of content-based algorithms as the network grows. They suggest that such a methodology will even help gain the initial trust of the users on the recommender system. The approaches proposed in this paper involve existing methods and new methods but it does not address the issue of personalization. The same algorithms are applied for all users. It is mentioned that some users prefer content-based approach while others prefer relationship-based but the paper does not address the issue of identifying which approach will work best for a particular user.

Hsu et al. [7] propose a recommender system for LiveJournal, which is a blog service community network consisting of single directional links between users. The authors use a hybrid of collaborative, structural and content-based approaches. By analyzing the available features from LiveJournal, the authors construct the features for learning the concept of recommendation. For a recommendation of link u-v, the features used in learning are: (1) In-degree of u (popularity of the user, u), (2) In-degree of v (popularity of the user, v), (3) Out-degree of u (number of other friends besides the candidate; saturation of friends list), (4) Out-degree of v (number of existing friends of the candidate besides the user; correlates loosely with likelihood of a reciprocal link), (5) Number of mutual friends of u and v, (6) "Forward delete distance": minimum alternative distance from u to v, (7) Backward distance from v to u, (8) Number of mutual interests of u and v: $(N_u \cap N_v)$, (9) Number of interests of u: N_u, (10) Number of interests of v: N_v, (11) $(N_u \cap N_v)/ N_u$, (12) $(N_u \cap N_v)/ N_v$. Properties 1 to 7 are considered structural features whereas properties 8 to 12 are interest-based features. Logistic regression is

used as the learning algorithm. The authors inferred that using all the features for learning produces better result than using one or a subset of the features. The approach described in this paper is very close to our research in terms of the data used for the recommendation but differs in the way our proposed system generates the recommendations. We use the recommendation strategies to compile the list of recommended candidates and use learning to rank algorithm to rank the candidates based on historic activity data of the user.

Tang et al. [23] propose a system for topic-based social network search for influential users. The algorithm works in three steps. In the first step, the topic in the query is broken into subtopics and users for the sub-topic are searched. In the second step, the influential users are identified from the result of the first step using a combination of language model and ACT (Author-Conference-Topic) model. The model considers the probability of the objects being associated with each topic using Gibs sampling and three matrices: $A*T$ (author by topic) count matrix, $T* V$ (topic by word) count matrix and $T* C$ (topic by conference) count matrix. The probability is calculated using these three matrices. The authors utilized the affinity propagation for social influence identification and construct a directed social influence graph for a particular topic. In the third step, a sub network connecting the influential users is discovered for top-k users of each topic. An approximate influence maximization-based algorithm is used which uses node's degree as a factor to selecting nodes. This approach only addresses the issue of finding topic-based influential user; however a user might be interested in finding new users for other reasons as well such as most popular user irrespective of topic, users in the same networking sharing many mutual friends and so on.

2.2.2 Recommender systems for Twitter

In this sub-section, we will review some of the recommender systems proposed for Twitter by the researchers in the past.

Hannon et al. [3] propose a recommender system called Twittomender, which is implemented using 9 different recommendation strategies. Strategy 1 uses user's tweets; strategy 2 uses followees' tweets; strategy 3 uses followers' tweets; strategy 4 is a hybrid combination of strategies 1, 2 and 3; strategy 5 uses followees' ids; strategy 6 uses followers' ids; strategy 7 is a hybrid combination of 5 and 6; strategy 8 uses a scoring function based on a combination of strategies 1 and 6; strategy 9 uses a scoring function based on the ranking of the user in each of the other recommendation lists. The top ranked common recommended users in the other strategies are selected. Strategies 1, 2, 3 and 4 use content-based approach, while strategies 5, 6 and 7 use collaborative filtering approach. Strategies 8 and 9 are hybrid ensemble. Based on their evaluation, it is found that strategies 8 and 9 provided the best results. The strategies used in this approaches are applied to all users to generate individual recommendations, hence, no personalization is applied here.

Chen et al. [24] propose a system to recommend URLs in Twitter using 12 different combinations of algorithms. Recommendation is generated using a three step process. First, a candidate URL is selected using either (1) URLs posted by followee and followee-of-followee (FoF) or (2) popular URLs in Twitter. Second, the content of the URL is ranked by topic relevance by either using self-profile or using followee-profile or not using topic relevance at all. Topic relevance is measured by calculating the term frequency inverse user frequency (TF-IDF) of the content. Frequent words weigh more; however, if the word is used by many other users,

i.e. if it is a commonly used word, it will weigh less. TF measures the frequency of the word in the user's tweets whereas IDF measures the percentage of users using this word. The authors choose to weigh people based on the frequency of their tweets, the higher the frequency of tweet the lower the weight. Third, social voting is considered. There are again two options: (1) voting, (2) no voting. It is found from their evaluation that the combination of FoF, self-topic and social voting provided the best result. When using the combination of all three approaches, the system generates the candidate list using FoF and ranks the list based on the product of the self-topic and social voting scores. The model proposed by the authors in this paper is applied to all users in the same manner without any personalization. In addition, the model only uses URLs from followee of followee and does not taken into account URLs from follower or followers of followers, or followers of followee or followee of followers or from any similar users.

Pennacchiotti and Popescu [25] talk about how user classification can be used for finding authority users, re-ranking in web searching and user recommendation in Twitter. User classification is done using values of attributes such as age, location, gender, ethnicity, origin, etc. However, values of these attributes are not always correct or filled and thus the researchers experiment with automatic methods that can populate the values of these attributes. The authors propose a machine learning approach to classify user based on the user-centric values (such as user's tweets, social behavior and interaction) and social graph information. Classification is done using the machine learning model using the user-centric information and then refined with the classifications of the neighbors using social graph information. User-centric information is found more valuable than social graph information for user classification. Though this paper talks only about user classification, the information provided was helpful for our research as we

use the user-centric information to find similar users. Finding similar users is one of the many strategies we use for our proposed recommender system.

Lappas and Gunopulos [26] propose an explainable interactive recommender system by using social endorsement. Endorsement is the act of approving a content or user by another user. In Facebook, users endorse by "liking", whereas, in Twitter, users endorse a user by following him/her. Social Endorsement Network can be defined as a bipartite graph (one-way) $G = (U, V, E)$ where U is the set of users, V is the set of entities and E is the set of endorsement edges. Tagging is used to understand the values of the different attributes of the endorsement content. The aim is to understand what the users like and what similar users like and use it for recommendation. The authors use endorsement network to understand why a user endorsed a particular item, i.e. provide explanation for the previous choice and use it to search or recommend for new items. This is known as the explainable recommendation. The authors propose an interactive recommender system where user can search for users using the endorsement graph or search for preferred interesting items of common endorsers. In their experiment, they tested their proposed approach with data from Twitter. The approach proposed in this paper uses only the following relationships to reveal endorsement. In our approach, besides using the following relationships, we use retweet relationship as an indication of endorsement.

Weng et al. in [27] proposes a system, TwitterRank, to find topic-sensitive influential users in Twitter. Currently Twitter measures the popularity or influence of a user by the number of followers. It is assumed that if many people follow a user, the user's tweet is read by these large number of followers and thus he/she is popular in his/her network. Weng et al. use similarity between users and PageRank using the graph structure to extract influential users for

their recommender system. User similarity is calculated based on topic similarity by analyzing contents of the tweets. Homophily is also a measuring factor for similarity. Homophily refers to the reciprocal relationship between two users. Since both users are interested in each other, they follow each other. PageRank is calculated using the topic similarity and the link structure of the network. Identifying influential users help in ranking of the search result and in marketing [27]. The approach used in this paper forms a subset of the strategies we use in our proposed approach. User similarity and PageRank are two of the many strategies our proposed approach use to recommend whom to follow.

Thonhauser et al. [28] propose a recommender system for Twitter using the novel concept of "Thought Bubbles". The authors refer to "Thought Bubble" as a topic-based interest that can be shared by many users. A user can be part of more than one thought bubble at any time and the association with any one bubble may change over time. The system generates recommendation based on the fact that users who belong to the same thought bubble are similar and they might be interested on following each other. First, the system filters the following-connections based on tweets and personal interest and then the outer connections to these filtered following-connections are used to identify possible interesting users. Second, Support Vector Machines and Natural Language Process techniques are used to classify the users followed by cosine similarity to rate the similarity. Other factors such as tweet frequency, follower ratio, amount of retweet and users' previous ratings are also considered. These factors are considered as ratios for the recommendation algorithms and are assigned different weights. The authors mentioned that the overall recommendation depends on the appropriate weighting scheme of these ratios. The approach described in this paper is similar to ours in the way that our approach uses many of the input information described here; however, we instead of assigning weights to

the different features, we use learning to rank methods to calculate the final score. In addition, instead of using the same weights for all users, our learning to rank methods uses historic followee-adding pattern of the user and produces a model that is already personalized for the specific user.

Tavakolifard et al. [29] propose a recommender system that first identifies the hidden web of trust and then proposes new friendship links for the expansion of network. The authors use the term "hidden web" to describe the subset of friends with whom the user interacts most frequently. The authors argue that a user may follow many other users but he/she may not necessarily interact with all of them frequently. The system works in two folds. In the first step, it identifies the most frequent interacting followees based on retweets, favourite tweets, and frequency of tweets. In the second step, the system proposes new links based on four propagation algorithms. The algorithms are simple-transitivity, weighted-transitivity, golbeck-transitivity and structural-similarity. Through evaluation using triadic closure, coverage, and MAE (Mean Absolute Error), it was found that structural-similarity algorithm provides the best result. The approach to identify the "hidden web" in this paper is similar to one of our recommendation strategies where we use the retweets and PageRank to reveal the most active user in the network; however, we use PageRank and we not only consider the user's retweets but also the retweets of the close network. In addition, structural similarity algorithm is one of the many recommendation algorithms we consider in our approach.

AlMeshary and Abhari [30] propose a system to recommend followees to Twitter users who recently moved to a new place. When people move to a new place/country where the language of communication is different from that where they came from, recommender systems cannot analyze any of their previous tweets to extract user's interest because of the language

difference. The authors address this problem by proposing a system that uses Natural Language Processing and Fuzzy Set concept to first translate the historical tweets and hashtags of the user and then use the extracted interest to find similar users in the new location. The system uses geo-location of the tweets in order to recommend followees belonging to the new location. Our approach differs in the way that it does not consider geo-location of the users for recommendation purpose but the system is flexible to include any new strategy in future. We consider a number of strategies and aims at building a personalized recommender system that will use the strategy or combination of strategies that have worked for this user in the past.

Silva et al. [31] suggest improvements to the currently used recommender systems by proposing ProfileRank. ProfileRank is a modified version of PageRank that uses content to rank the users and tweets instead of the structure of the social network. In this approach, the tweets and retweets are considered as links for the PageRank. The authors claimed that ProfileRank generates results better than that of traditional PageRank. The approach considered in this paper is similar to the one of many strategies we consider for our proposed recommender system. However, we do not use contents of the tweets for PageRank, instead we use retweet links for PageRank.

Celebi and Uskudarli [32] propose a recommender system that uses content extracted from tweets and retweets of the user. The system starts with extraction of words, hashtags, mentions and urls from the tweets and retweets of the user. It then undergoes a series of process to clean and process the content by removing stop words. Using TF-IDF scheme, it then searches for similar users in the network followed by calculating relevancy scores for the searched results. The relevancy scores consist of five different scores: Socialness, Feedness, HashtagUsage, Retweeted, and TermVariation. Socialness measures the connectedness of the user with others,

Feedness measures the amount of URLs in the content of the user, HashtagUsage measures the usage of hashtags, Retweeted measures the usage of retweets, and TermVariation measures the variation of words in the content. One set of these scores is calculated for the user and another set is calculated for the searched user making a total of ten scores. The final score is the linear combination of these ten scores with different weights. To increase effectiveness, minimum threshold values of individual scores are defined. In this approach, only content is used and structural links such as follower or followee relationships are not used. Our proposed approach also uses content from tweets and retweets to calculate similarity score and we also calculate individual scores from different recommendation strategies; however, we do not linearly combine the scores using different weights. We use learning to rank algorithms to generate the final score.

Armentano et al. [33] propose a recommender system that recommends candidates who share common followees with the target user and the candidate recommendation list is ranked based on the popularity based on most-mentioned score and PageRank score. The approaches mentioned in this paper match with some of the recommendation strategies of our recommended system; however, there are a few differences. The research described in this paper uses only one strategy to generate candidate list. In our research, we use seven different strategies to generate candidate list and scores. In their research, they use different strategies for candidate generation and ranking process. In our research, we use the same strategies for candidate generation and ranking process. Moreover, we use learning to rank algorithm in combination of user's historic data to choose which strategy would perform best for a particular user.

Garcia et al. [34] propose a recommender system that uses user's preference of popularity and activity to search for new followees. Popularity is defined as the follower to followee count

ratio, while activity is defined as the fraction of followees who are active. The user's popularity score and activity score is used as the threshold preference to search for users whose popularity and activity scores are above the threshold values. It is found that using both the popularity score and activity score generates better result than using only either popularity score or activity score. This paper was helpful in understanding that popularity is a contributing factor in generating candidate list for recommendation. We used popularity as one of the recommendation strategies in our proposed system; however, we did not use the same process that was used to measure popularity in this paper. Instead, we used PageRank of retweet and most mentioned count to calculate popularity. In our research, we not only use more strategies for the recommendation process but also use learning to rank algorithms to rank the candidate list.

Golder et al. [35] use the recommendation strategies of finding Followee-of-Followers and Followees-of-Followees to search for whom to follow in Twitter. The authors claim that since the user and the recommended candidate shares the same group of audience or followers, they share similar interests. In addition, followees of followees, whose tweets are retweeted often by the user's followees, are also considered as recommended candidates. The recommendation strategies mentioned in this paper matches with two of the recommendation strategies used in our research; however, we used in total of seven recommendation strategies. Moreover, we used learning to rank algorithm to rank the candidate list.

Gupta et al. [6] discuss the architecture of Twitter's recommender system, "Who To Follow", which was built in 2010 and was designed to run in a single machine using the users following relationships. The recommender system generates the recommendation using a variety of recommendation algorithms and Twitter's in-memory graph processing engine, Cassovary. Among the recommendation algorithms used are: identifying authority users using SALSA,

finding similarity using following relationship, finding most common neighbors, finding closure and many others. This first generation recommender system of Twitter suffers from scalability bottleneck. With the growth of Twitter, there will be more users and more edges to consider. The in-memory graph processing approach will not be sufficient to hold the entire new graph in memory. The approach described here is different from ours because this system is based solely on graph structure and does not use retweet, tweet, mention and reply information, which form the basis of content-based recommendation. The paper is focusing more on their system architecture instead of the details of the recommendation algorithm. Some of the technical details are missing for a better understanding on Twitter's current recommender system.

Gupta et al. [6] also briefly mention that to overcome the limitations, they are currently working on their second generation recommender system that will use the Hadoop distributed framework to support the Twitter graph. This improved recommender system will be using user interactions such as retweet, mentions and favorites in addition to the graph structure. Candidate recommendation list will be generated using various algorithms and the scores in the recommendation will be rescored with the help of machine learning using logistic regression classifiers. Little has been mentioned in this paper about the details of this new system as it is currently under development. Personalization is mentioned to be a characteristic of their new recommender system but it is not mentioned if the ranking model will be personalized for each user.

2.3 Summary

In this chapter we have reviewed some of the concepts used in the rest of this thesis and also discussed the related work done in the area of recommender system for Twitter and for other social networking sites. From these reviews, we can see that most of the proposed recommender systems [5][3][24] use many different information; however, the recommendation algorithms are used for all users. While a recommender system can generate a very good recommendation using collaborative-based approach, the user may be interested in the results from content-based approach. Chen et al. mentions in [24] that some user may prefer "relevance" while others may prefer "serendipity". Finding the most popular user, the most similar user or the closest person based on mutual following relationship is not always the solution for recommender systems. A strategy may work for one user but it may not work for all users. Recommender systems need to be trained on the users' past following-adding pattern or following preference. There is a need for the recommender system to be personalized. More research is needed in the area of ranking and summarization. With the overloading of information, recommender systems need to be tuned both for proper filtering and accurate recommendations.

CHAPTER 3
PERSONALIZED RECOMMENDER SYSTEM ON WHOM TO FOLLOW IN TWITTER

As discussed in the previous chapters, we propose a recommender system for Twitter using the historic activity pattern of the users. We extract related data from Twitter using Twitter API. Several recommendation scoring algorithms are then applied followed by learning to rank methods. In the following sections, we explain the system architecture, historic data required for the scoring and ranking process, the recommendation algorithms for scoring and the learning to rank methods.

3.1 System Architecture

In this section, we will explain the overall architecture of the main components of our system which was used to recommend new followees to the users. Since we will not be relying on a single strategy, we will utilize a series of strategies followed by machine learning. Here we discuss each of the components in detail and give reasoning as to why such method was used. We will also describe each of the strategies in detail along with their algorithms.

The architectural model of our proposed recommender system is shown in Figure 1, which consists of two parts: the applications and the storages.

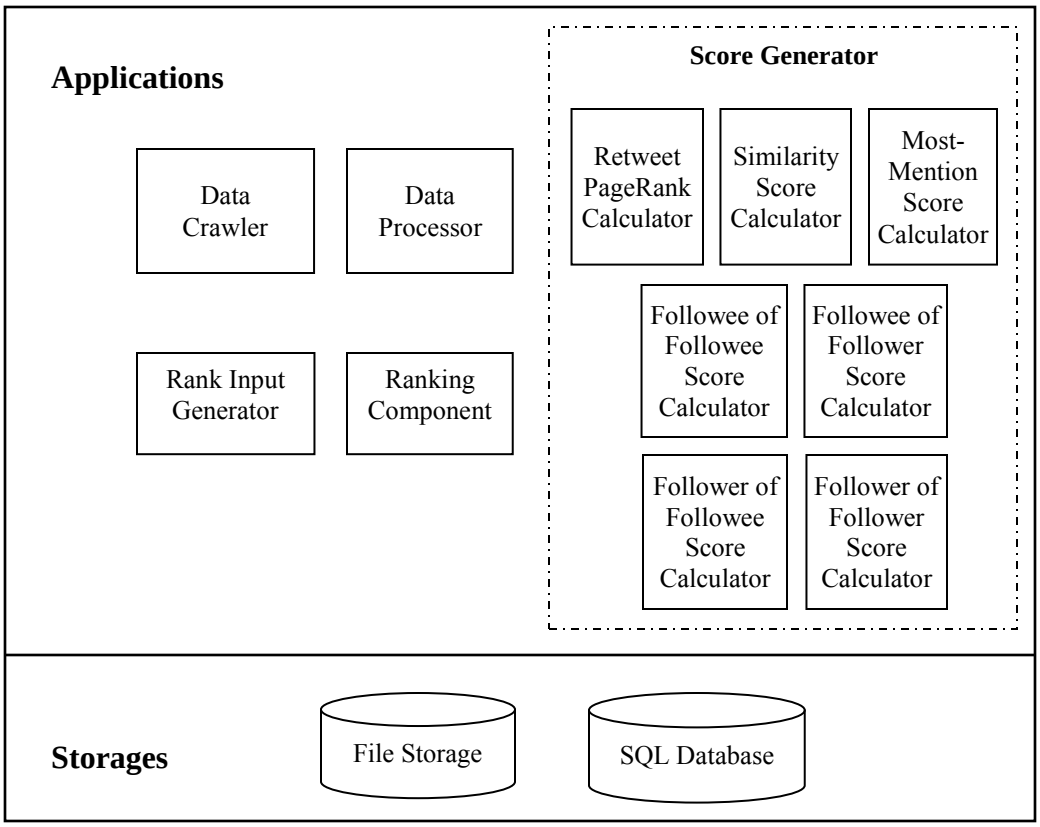

Figure 1: Architecture of our recommender system

Among the application components, we have the Data Crawler, Data Processor, Score Generator, Rank Input Generator and Ranking Component.

Data Crawler is the application to collect the data required for our recommender system. It is responsible to retrieve the social network data that will be used by our recommender system to generate recommendations. This application makes the required http calls to the Twitter API to request specific data. Among the data requested is the basic profile information of the user, the list of followers of the user, the list of followees of the user, the tweets of the user, tweets of others that have been retweeted by the user and the list of other users that the user mentioned in her tweets. The data crawler application needs a list of user ids, seeds, whose data needs to be

collected. The data collected by the crawler is saved in text files located in the file storage directory. Since the crawler will be running in multiple machines simultaneously and since we wanted to avoid setting up local databases in those machines, we have decided to configure the crawler to download the collected data in text. The crawler is scheduled to run in certain intervals of time so that we will have different sets of the user's data at different times. This is essential for our recommender system as it utilizes the historic activity pattern of the user.

Data Processor is the second application that will be triggered once our data crawler completes collection of data. The text files containing all the collected data is made available to the data processor which then combines the data from the text file and prepares the data to be stored in the SQL Database. The process of preparing the data consists of combining the data from text files, cleaning the data, removing extra spaces and special characters, generate SQL insert statements for multiple insertion and saving the output in a SQL file. The processor then executes the SQL file and the data gets added to the SQL database. We have made our system as modular as possible so that each step can be executed separately independent of each other. By saving the raw collected data in text files, we have avoided installing local databases in every machine and have avoided using a server database. Rather than triggering the Data processor to run for every machine separately, we have made the data available to the processor in one package and then run the process for all data at the same time. Generating a SQL file is efficient for us because it allows multiple-insertion which is the optimized and efficient way to add records in the database. In addition, the generated SQL file gives us the option to re-enter the data in the SQL Database in the event of data loss or data corruption from the database.

The combined responsibility of the data crawler and the data processor is to make sure that we have made ourselves available to a subset of the Twitter data that will be sufficient for

our research experiment. Since our recommender system is not residing on Twitter and do not have direct access to Twitter data repository, we use our data crawler and data processor to download the required data and save it in our local database that can be used offline anytime by our recommender system.

The Score Generator is the main component of our recommender system. It is here that the scores based on different recommendation strategies are generated using social network data from the SQL Database. The output scores are then saved in the SQL Database. Currently we have seven sub-modules, each of which corresponds to a single recommendation strategy. The sub-models are listed below:

1. Retweet PageRank Score Calculator
2. Similarity Score Calculator
3. Most Mention Score Calculator
4. Followee-of-Followee Score Calculator
5. Followee-of-Follower Score Calculator
6. Follower-of-Followee Score Calculator
7. Follower-of-Follower Score Calculator

Each of the modules is responsible to generate the list of candidate recommendations along the individual scores based on its recommendation strategies. The scores will show the importance of the recommendation. The recommendation strategies used in each of the modules of Score Generator will be discussed in detail in the next section. Though we currently use seven different strategies, the system is flexible to include more recommendation strategies in the future.

Rank Input Generator is the next module that gets executed after the scores from the Score Generator become available. The Rank Input Generator gets the list of candidate recommendations along with the scores for each of the recommendation strategies used in the earlier modules. It creates a final list of candidate recommendations by combining the lists from each of the different recommendation strategies. It is responsible to generate a file to be used as input to the Ranking Component. This generated file consists of the list of candidate recommendations, their individual scores from all different strategies and a relevancy score that indicates the chance of the user actually accepting this recommendation. One of the contributions of our recommender system is its use of user's historic followee adding pattern. The relevancy score mentioned earlier indicates if the candidate recommended was added as a followee by the user.

The final module of our system is the Ranking Component. It consists mainly of learning to rank modules. It makes use of the input file generated in the earlier step and employs an evaluation metric to aid in the process of generating a ranking model. This ranking model generated using the scores of different recommendation strategies from the historic activity data of the user is a complete ranking model personalized for this user.

3.2 Recommendation Strategies

Recommendation strategies are the different ways a user finds someone interesting and adds him/her as a new followee. We assume that a single user may use any of the strategies mentioned below or a combination of these strategies in order to search for a new followee. In reality, people use many more strategies but due to the limitation of our research, we only consider these few strategies. The user may or may not be aware of the strategies he/she uses to

search for whom to follow. Recommendation strategies calculate scores for the recommended candidates. Since recommendation strategies are just different ways a followee can be searched and added, the scores obtained from the strategies represent the likelihood of finding that recommended followee. Since for each of our recommendations, we will have scores from seven different strategies, we will be able to find the probability in seven different ways. The aim of the strategies is to rank the recommendations using different reasons of importance. For a recommendation, these scores will reveal attributes such as how important the candidate is in the social network, how close the candidate is to the target user in the network, whether it is possible that the target user already know the candidate in real life and so on. The sub-sections below describe each of the strategies in detail.

In the following discussion we will use notations u to represent the target user to whom we are offering the recommendation, u_i to represent the recommended candidate user and v_j to represent a followee or follower of the target user u.

3.2.1 PageRank with Retweet Strategy

This strategy returns a ranked list of users along with their authority scores. A higher authority score represents higher popularity in the social network. They can be considered as leaders or authority figures as many others follow them and consider their advice useful. Researchers previously used the number of followers and/or followees and their links to calculate authority users using PageRank algorithm [27]. However, in our research, we used retweets for PageRank calculation. There are users who are followed by many others but they may not necessarily have lots of input/advice in the social network. For example, a celebrity may be very famous for the work he/she does and he/she may be followed by many users, but he/she may not have much input in the social network. Tweeting is considered a contribution to the

social network because this is the way a user can share valuable information with his/her followers. If he/she is not an active or contributing user, then it not worth recommending him/her to other users. Retweet, on the other hand, is a clear measure of popularity and contribution endorsed by others in the social network. Retweet is the act of sharing others' tweets. It is a way of expressing agreement to the opinion stated in the tweet by the original author of the tweet. While tweets show the activity of the users, retweets show that other users are in favor of the tweets. A higher number of retweets represents a higher acceptance of the opinion expressed in the tweet.

Since this strategy will use retweets from the entire dataset, it does not need to be recalculated for each user, rather one output ranked list can be used to generate recommendation for any user. The process starts with getting the list of all users from the social network dataset repository. For each of the users, a list of retweets is then retrieved from the dataset. These retweets have information of the original author. Since the user is found to retweet tweets of the original authors, there is a valid accepted close connection between this user and the original author. We consider this connection as the retweet link. Retweets links for all the retweets done by the user is used to form the retweet graph.

The process is repeated until we consider all the users in the entire dataset, which will result in the completion of our retweet graph. Links of this graph are then used as input for our PageRank calculation as described in Section 2.1.1. The result of the PageRank calculation will be a sorted list of users in the dataset along with their ranking score.

The recommendation algorithm (pseudo code) for strategy 1 is given below. The first algorithm illustrates how the input for the PageRank is composed and the second algorithm illustrates the core functionality of this strategy.

Algorithm 1: Pseudo code to compose input for the PageRank

```
TextFile Compose-PageRank-Input( ) {

        pageRankInputFile = Create new TextFile
        userList = get list of all users from database
        for each user in userList {
                userID = get ID of user
                retweetList = get list of all retweets of userID
                for each retweet in retweetList{
                        authorUserID = Find original author ID of retweet
                        retweetLink = Create link between userID and authorUserID
                        Save retweetLink to pageRankInputFile
                }
        }
        return pageRankInputFile
}
```

Algorithm 2: Pseudo code for the PageRank with Retweet Strategy

```
void Calculate-Retweet-PageRank( ) {
        inputFile = composePageRankInput ( )
        //Compute PageRank based on formula 2.1, 2.2 and 2.3
        ranking-scores = PageRank(inputFile)
        save ranking-scores in database
}
```

3.2.2 Similarity Strategy

This recommendation strategy finds other users similar to this user. The strategy uses the similarity of the contents of the tweets as a measure to determine the similarity of users. Each tweet consists of maximum 140 characters. Although the length of the content of a tweet is very short, each tweet consists of the user's message. It reveals a message belonging to a particular topic. Analyzing tweets, we can determine the topics that the user is interested in. The strategy extracts words from the user's tweets and forms a document. The document is then used to compare documents of other users. Each of these documents contains the words used by the users in writing the tweets and studying these words the system can identify the key-words that are used most often. In order to avoid confusion and be more focused on the actual topic of the tweets, the very common words called the stop words are eliminated in this comparison process. These documents not only reveal the topics of interest, but also rank the topics based on the frequency of occurrence of the keywords in the documents. User's interest on a particular topic may change over time. Users of social network get involved in different topics of discussion and this strategy finds similar users based on recent topics of discussion. Though the data repository may warehouse thousands of tweets, it is not worth considering all of them. A threshold needs to be defined and the strategy should focus on finding similarity based on that certain number of the user's most recent tweets.

This strategy utilizes Vector Space Model (VSM) to calculate the user similarity using the popular similarity technique, Cosine Similarity. As mentioned earlier in Section 2.1.2, Cosine Similarity measures the cosine of the angle between the two vectors. Each of these vectors is constructed using the document containing all the tweets from one user. Since the algorithm compares two users, we have two documents, each representing one user. These are weighted

vectors because the frequencies of the words in the documents are considered in the term vector units. Finally the Cosine Similarity is calculated using Formula 2.4. The procedure to calculate scores using the similarity strategy is illustrated in Algorithm 3.

Algorithm 3: Pseudo code for Similarity Strategy

```
void Calculate-Similarity( ) {
        numUser = Get total number of users in the dataset
        users = Get list of users in the dataset

        for (int i = 0; i < numUser; i++) {
                tweets = Get tweets of user[i]
                Compose Document[i] using tweets of user[i]
                Create vector[i] of Document[i]

        }

        scoreList = create new score list

        for (int i = 0; i < numUser; i++)
        {
                for (int j = i+1; j < numUser; j++)
                {
                        similarity-score = Calculate Similarity Score (vector[i], vector[j])
                                        using Formula 2.4
                        Save score in scoreList

                }

                find top k similar users from scoreList
                save similarity-score in Database

        }

}
```

The result of the similarity strategy will be a ranked list of similar users along with the individual scores.

3.2.3 Most Mentions Strategy

Most Mention Strategy is another popular recommender strategy similar to PageRank Strategy mentioned in 3.2.1. In this recommendation strategy, recommendation is generated based on the popularity of the users in the social network graph. Mention of a user in other user's tweets and retweets is used for candidate selection. This strategy returns a list of ranked candidate users where the ranking is done based on the frequency of mention of the candidate in others' tweets and retweets. Mentioning a user in tweets and retweets is a good representation of the popularity of the user. When a user is mentioned often in tweets and retweets, the user is considered to be a famous and important person in the social network as he/she is involved in the interaction of other users. A user in Twitter can follow many other users and can also be followed by many other users; however, if he/she is not involved in the interactions, he/she participation is not active. One of the aims of social recommender systems is to recommend and promote active users. If a user is mentioned in other users' tweets, it is a clear indication that the mentioned user's presence is required in the conversation or broadcast of the tweet. Whether it is to reply to the mentioned user or to attract the attention of the mentioned user, mentioning a user represents the popularity of the user in the social interaction. Most mentioned score of the candidate user u_i who is recommended to the target user u as a potential followee is calculated based on Equation (3.3).

$$MS(u_i) = \sum_{u_k \in U} MC(u_k) \tag{3.3}$$

Where,
$MC(u_k) = the\ number\ of\ mentions\ of\ user\ u_i$ from user u_k
U = set of all users
$u_i \notin Fe(u)$
$Fe(u) = set\ of\ Followees\ of\ u$

Most Mention strategy calculates the candidate scores using the contents from the tweets of all users in the network. The algorithm calculates frequency of mention of the candidate user in the tweets of all other users in the network. It outputs a ranked list of recommended users along with the ranking score. Its pseudo code is listed in Algorithm 4.

Algorithm 4: Pseudo code for Most Mention Strategy

```
void Calculate-Most-Mention( ) {
        numUsers = Get number of all users in the dataset
        users = Get list of all users from dataset
        allMentions = create empty list of mentioned user

        for (int i = 0; i < numUsers; i++) {
                tweets = Get tweets of the users[i]
                mentions = Extract mentioned users from tweets
                //add mentions in allMentions
                allMentions.add(mentions)
        }

        //create empty 2-dimensional list of mentioned user and associated score
        mentionScore = new list (2)

        //calculate most mention strategy scores by extracting unique mentioned user from allMentions
        // and counting the occurrence of each mentioned user in allMentions
        mentionScore = Calculate-most-mention-strategy-scores(allMentions)

        Save mentionScore in Database
}
```

3.2.4 Followee of Followee Strategy

Followee-of-Followee recommendation strategy recommends users who are followed by many of user's followees. Followees are defined as other users whom the user follows. Tweets and retweets of the followee are visible to the user. The idea here is if the user follows a number of other users (followees) and many of these other users (followees) follow one common user

(followee-of-followee), then this user might be interested in following that user (followee-of-followee) directly. Figure 2 shows an example where user *A* represents the target user for whom the system is generating recommendation. User *A* follows User *F*. The arrows show the direction of the flow of information. Since *A* follows *F*, information from *F* is accessible to *A*. User *F*, on the other hand, follows User *B*. Since information flows from *B* to *F* and from F to *A*, User *A* might be interested in following User *B* directly. In this example, *B* is a followee-of-followee of *A*. The user is already getting the data feed from the followees. However, if the information useful to the user is coming from the followee-of-followee and getting it filtered and broadcasted (retweeted) by the followees, then the user might as well be connected to the followee-of-followee, who is a source of information. The aim is to reduce the distance between the user and the source of information.

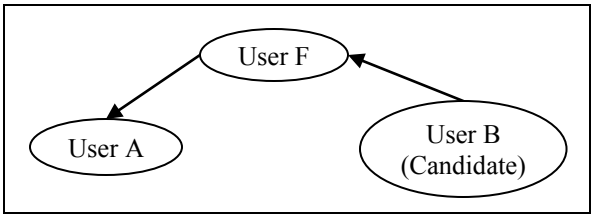

Figure 2: Followee-of-Followee social network graph showing flow of information

If such a followee-of-followee exists from whom many of user's followees are getting data and retweeting the information, the user might be getting lots of duplicate data. Since the information is coming from a single source to the followees of the users who in return broadcast the information, the user is certainly getting duplicated information. Following the followee-of-followee is one solution to solve getting these duplicate tweets followed by the removal of existing followees. Our recommender system does not propose any removal recommendation. It is up to the user to decide how she wants to make her following list clean and precise.

Followee-of-Followee recommendation strategy is also useful in finding users with whom the user is friends in real life. If the recommended user is a mutual friend of the followees of the user and if the user is friends with these followees in real life, chances are the user knows this followee-of-followee as well.

This recommendation strategy is based solely on the structural information of the social network graph. The algorithm collects related data from the social network repository. Data includes list of followees and list of followees-of-followees. The system determines the followees-of-followees who are not yet followees of the user and calculates a score based on their frequency of occurrence in the followee list. The score is calculated based on Formula 3.4, where *FeFe(u_i)* is the Followee-of-Followee score of the candidate user u_i and it calculates the number of followees (of the target user u) who are followers of the candidate user. Since the candidate user will be recommended as a potential followee of the target user, any existing followees of the target user is exempted from the candidate list. The strategy outputs a ranked list of followees-of-followees. The pseudo-code is shown in Algorithm 5.

$$FeFe(u_i) = \sum_{\forall v_j \in Fe(u), v_j \neq u_i} Followee(u_i, v_j) \tag{3.4}$$

Where,
$u_i \notin Fe(u)$
$Fe(u) = set\ of\ Followees\ of\ u$
$Fe(v_j) = set\ of\ Followees\ of\ v_j$
$$Followee(u_i, v_j) = \begin{cases} 1, & u_i \in Fe(v_j) \\ 0, & u_i \notin Fe(v_j) \end{cases}$$

Algorithm 5: Pseudo code for Followee-of-Followee Strategy

```
void Followee-of-Followee ( ) {
        followees = Get list of followees of the target user
        numFollowees = Get number of followees
        candidate-pool = create new list

        //create 2-dimensional list for candidate userID and associated score
        followee-of-followee-score = create new list

        for (int i = 0; i < numFollowees; i++ ) {
                followee-of-followees = Get followees of followees[i]
                for each candidate-user in followee-of-followees{
                        //if the candidate-user is not already a followee of the target user
                        if candidate-user is not in followees{
                                add candidate-user to candidate-pool
                        }
                }
        }

        //enter distinct candidate user from candidate-pool
        followee-of-followee-score.addCandidates(candidate-pool)

        //calculate candidate score based on the number of occurrences of the candidate
        //     in the candidate-pool
        followee-of-followee-score.addScore(candidate-pool)

        Save followee-of-followee-score in database
}
```

3.2.5 Followee-of-Follower Strategy

Similar to the strategy mentioned in Section 3.2.4, this Followee-of-Follower strategy generates recommendation based on the followee-following relationship. A followee is another user whom the target user follows and a follower is another user who follows the target user. Social networking information flows from the followee to the user and then from the user to the follower. This strategy generates recommendation based on the fact that if many followers of the user also follow another user, there must be some similarities between that user and this target user. In Twitter the users are connected by single-directional links. A user may follow another

user but the second user may not follow back the first user. All the followers who follow the user are getting tweets from the user. They either find the user's information useful or interesting. Whatever the reason for following is, these followers want to follow the user. If a group of these followers, who follow the user, also follow another user, then these followers find both the first user and the second user interesting. In Figure 3, if the target user is *A* and the followee-of-follower is *B*, then the follower of *A, F*, finds both *A* and *B* useful. If *F* finds the combination of information from *A* and *B* useful, *A* is missing something that is broadcasted by *B*. This strategy aims at identifying the useful *B* for *A*. The arrows in Figure 3 show the direction of flow of information in the network. The user and followee-of-follower may share common interests, may be friends in real life, may know each other personally, may work at the same place, may live at the same location, may buy products or services from the same company, and so on. Thus the user might be interested in something her followers are interested in.

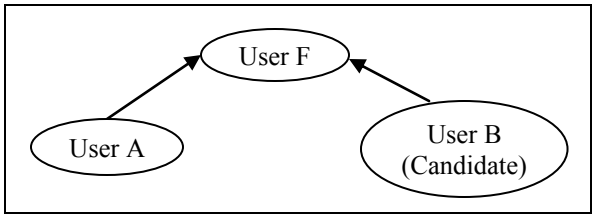

Figure 3: Followee-of-Follower graph structure showing flow of information

The recommendation of this strategy is based on the structural information of the social network graph. The algorithm collects related data from the social network repository. Data includes list of followers and list of followees of followers. The system determines the followees of followers who are not yet followees of the target user and calculates a score based on their frequency of occurrences of these followees-of-followers in the followers' list. The strategy outputs a ranked list of followees-of-followers. The score is calculated based on Formula 3.5,

where $FeFr(u_i,u)$ is the Followee-of-Follower score and it calculates the number of mutual followers of the target user u and the candidate user u_i. Since the candidate user is recommended as a potential followee of the target user, any existing followees of the target user are exempted from the candidate list. The pseudo-code is shown in Algorithm 6.

$$FeFr(u_i, u) = \sum_{\forall v_j \in Fr(u), v_j \neq u_i} Followee(u_i, v_j) \tag{3.5}$$

Where,

$u_i \notin Fe(u)$

$Fe(u) = set\ of\ Followees\ of\ u$

$Fr(u) = set\ of\ Followers\ of\ u$

$$Followee(u_i, v_j) = \begin{cases} 1, & u_i \in Fe(v_j) \\ 0, & u_i \notin Fe(v_j) \end{cases}$$

Algorithm 6: Pseudo code for Followee-of-Follower Strategy

```
void Followee-of-Follower ( ) {
        followees = Get list of followees of the target user
        followers = Get list of followers of the target user
        numFollowers = Get number of followers
        candidate-pool = create empty list

        //create 2-dimensional list for candidate userID and associated score
        followee-of-follower-score = create new list

        for (int i = 0 ; i < numFollowers ; i++ ) {
                followee-of-follower = Get followee of followers[i]
                for each candidate-user in followee-of-followees{
                        //if the candidate-user is not already a followee of the target user
                        if candidate-user is not in followees{
                                add candidate-user to candidate-pool
                        }
                }
        }
        //enter distinct candidate user from candidate-pool
        followee-of-follower-score.addCandidates(candidate-pool)

        //calculate candidate score based on the number of occurrences of the candidate
        //     in the candidate-pool
        followee-of-follower-score.addScore(candidate-pool)
        Save followee-of-follower-score in database
}
```

3.2.6 Follower-of-Followee Strategy

Follower-of-Followee recommendation strategy is a structural-based algorithm and relies on the followee-following relationship of the user. It does not make use of any content information. In Figure 4, if user *A* is the target user, user *F* is a followee of user *A* and user *B* is a follower of user *F*, then user *B* is a follower-of-followee of user *A*. Since both *A* and *B* are interested in the same user *F*, they may share the same interest. They may or may not be part of the same topic of discussion. However, it is clear that the user shares some interest with the candidate user. The more the number of mutual followees of the user and the candidate user, the more the number of interests they share. This strategy aims at finding candidate users who follow the same set of users as this target user follows and then ranking them accordingly.

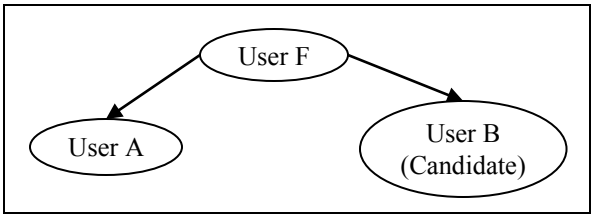

Figure 4: Follower-of-Followee graph structure showing flow of information

The algorithm for this strategy collects related data from the social network repository. Data includes list of followees of the users and list of followers of those followees. The system determines the followers of the followees who are not yet followees of the user and calculates a score based on their frequency of occurrences of these followers-of-followees. The strategy outputs a ranked list of followers-of-followees. The score is calculated based on Formula 3.6, where $FrFe(u_i,u)$ represents the follower-of-followee score and calculates the number of mutual followees of the target user u and the candidate user u_i. Since the candidate user is recommended

as a potential followee of the target user, any existing followees of the target user are exempted from the candidate list. Pseudo code is listed in Algorithm 7.

$$FrFe(u_i, u) = \sum_{\forall v_j \in Fe(u), v_j \neq u_i} Follower(u_i, v_j) \tag{3.6}$$

Where,
$u_i \notin Fe(u)$
$Fe(u) = set\ of\ Followees\ of\ u$
$Fr(v_j) = set\ of\ Followers\ of\ v_j$
$$Follower(u_i, v_j) = \begin{cases} 1, & u_i \in Fr(v_j) \\ 0, & u_i \notin Fr(v_j) \end{cases}$$

Algorithm 7: Pseudo code for Follower-of-Followee Strategy

```
void Follower-of-Followee ( ) {
        followees = Get list of followees of the target user
        numFollowees = Get number of followees
        candidate-pool = create empty list

        //create 2-dimensional list for candidate userID and associated score
        follower-of-followee-score = create new list

        for (int i = 0 ; i < numFollowees ; i++ ) {
                follower-of-followee = Get followers of followees[i]
                for each candidate-user in followee-of-followees{
                        //if the candidate-user is not already a followee of the target user
                        if candidate-user is not in followees{
                                add candidate-user to candidate-pool
                        }
                }
        }

        //enter distinct candidate user from candidate-pool
        follower-of-followee-score.addCandidates(candidate-pool)

        //calculate candidate score based on the number of occurrences of the candidate
        //     in the candidate-pool
        follower-of-followee-score.addScore(candidate-pool)

        Save follower-of-followee-score in database
}
```

3.2.7 Follower-of-Follower Strategy

This recommendation strategy aims at completing the formation of the inner circle of trust of the user. In Figure 5, if user *A* is the target user, user *F* is the follower of user *A* and user *B* is the follower of user *F*, then user *B* is the follower-of-follower of user *A*. User *A* is connected to the candidate user *B* via one of more follower, the user may know the candidate user in person. The recommendation strategy provides suggestion to the user if candidate user *B* is a person already known to the user or if the user is interested in following her. The candidate user *B* may already be getting tweets of the user that are retweeted by the follower, *F*, of user *A*. If the user knows the followers in real life, chances are she may also know the follower-of-follower and may want to get connected with the candidate user.

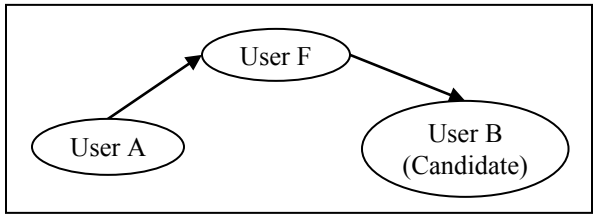

Figure 5: Follower-of-Follower graph structure showing flow on information

This recommendation strategy is based on the structural information of the social network graph. The algorithm collects related data from the social network repository. Data includes list of followers and list of followers of followers. The system determines the followers-of-followers who are not yet followees of the user and calculates a score based on their frequency of occurrences or number of links with the followers of the user. The strategy outputs a ranked list of followers-of-followers. The score is calculated using formula 3.7, where $FrFr(u_i,u)$ represents the follower-of-follower score and calculates the number of followers (of the target user u) who are followees of the candidate user u_i. Since the candidate user is recommended as a potential

followee of the target user, any existing followees of the target user are exempted from the candidate list. The pseudo code is shown in Algorithm 8.

$$FrFr(u_i, u) = \sum_{\forall v_j \in Fr(u), v_j \neq u_i} Follower(u_i, v_j) \tag{3.7}$$

where,
$u_i \notin Fe(u)$
$Fe(u) = set\ of\ Followees\ of\ u$
$Fr(v_j) = set\ of\ Followers\ of\ v_j$
$Follower(u_i, v_j) = \begin{cases} 1, & u_i \in Fr(v_j) \\ 0, & u_i \notin Fr(v_j) \end{cases}$

Algorithm 8: Pseudo code for Follower-of-Follower Strategy

```
void Follower-of-Follower ( … ) {
        followees = Get list of followers of the target user
        followers = Get list of followers of the target user
        numFollowers = Get number of followers
        candidate-pool = create empty list

        //create 2-dimensional list for candidate userID and associated score
        follower-of-follower-score = create new list

        for (int i = 0 ; i < numFollowers ; i++ ) {
                follower-of-follower = Get follower of followers[i]
                for each candidate-user in followee-of-followees{
                        //if the candidate-user is not already a followee of the target user
                        if candidate-user is not in followees{
                                add candidate-user to candidate-pool
                        }
                }
        }

        //enter distinct candidate user from candidate-pool
        follower-of-follower-score.addCandidates(candidate-pool)

        //calculate candidate score based on the number of occurrences of the candidate
        //     in the candidate-pool
        follower-of-follower-score.addScore(candidate-pool)

        Save follower-of-follower-score in database
}
```

Although we used only seven strategies for our proposed recommender system, the system is not limited to these strategies. The system is flexible for any number of strategies to be added in future. Different strategies are used to provide scores that represent importance level for different reasons. The system is modular and is able to incorporate any change to the existing strategies, removal of existing ones and addition of the new ones to the pool of strategies.

Rather than imposing a recommendation strategy to the users, we are offering the system a number of different recommendation strategies and letting the system decide which ones will be more suitable for the user based on the past following pattern from the user. We are aiming at generating a recommendation that the user will be more comfortable with as she might have followed the same strategy in the past.

3.3 Personalized Recommendation

As discussed in the previous sections, our proposed recommender system makes use of many strategies. Each of the strategies generate candidate list and calculates scores for the candidates. Each of the strategies mentioned earlier works differently. The strategies generate scores based on individual algorithms. Since the strategies use different data and calculate scores differently, they generate recommendations for fulfilment of different interests. Based on the scores from each of these strategies, the candidate recommendations are ranked. Since each person is unique and motivated differently, recommendations or strategies applied for one user may not work for another user. Some users are interested in finding people they already know while others may be adventurous in finding new friends. One person may even have two user accounts: one can be used for work and the other can be a personal one. Types of activities in the two accounts can be different.

One of the main contributions of our research is to propose a recommender system that is personalized for individual users. A recommender system that learns the type of followee the user is interested in and recommends new friends to follow based on the previous selection of followees. Our proposed recommender system utilizes all the strategies mentioned earlier, but we do not linearly combine the scores from the different strategies. Neither do we assign any pre-defined weights to the strategies nor do we select the strategies randomly. Instead of using one strategy or all strategies for all the users, we aim at identifying the strategy that the user used in the past. By choosing specific strategies for a user, we make the recommender system personalized for the user. The recommender system we propose in this thesis is personalized for each user.

Our proposed recommender system uses the user's historic activity data to find the right combination of strategies for the user. Scores of the strategies are calculated using the historic data and the recommendation is validated using a relevancy factor. The value of the relevancy factor is derived from the current condition and is an indicator if the recommended candidate is later added as a followee by the user. Our assumption is that the scores generated using yesterday's data can be effectively used to predict today's followees and similarly, today's condition can be analyzed to determine tomorrow's followees, etc.

To help the recommender system understand the pattern the user is following to add new followees to her profile, we use learning to rank algorithms. The learning to rank algorithm constructs a personalized model to rank the new recommendations for the user. The recommendation strategies are applied to the available historic data sets in order to build the training data for the learning to rank algorithm. In addition, the strategies are applied to the current data to get scores that are used to generate recommendations.

The first part of generating the recommendation is to generate the candidate list based on the scores from all different strategies. Next, we use learning to rank algorithm to rank the items in the candidate list. We used many learning to rank algorithms to find the algorithms that works best for our data. In general, learning to rank algorithms take in three inputs: training data, validation data, and test data. The algorithm learns and builds the model based on the training data; however, at every iteration, it validates the learning by the use of the validation data. Each learning to rank algorithm has its own input representation and loss function. Loss function, often called the cost function, is used in the learning process to measure the deviation of the predicted value from the actual value. Learning to rank algorithms aim at minimizing the output from the loss function, and thus optimize the learning process. Training data and validation data are used to build the model, whereas test data is used to measure the performance of the resulting model using an evaluation metric.

3.4 Summary

In this chapter we explain the methodology of our research in details. We start with a brief introduction of the architecture of our personalized recommender system and present description for each of the components comprising the proposed system. We explain the different strategies involved in generating recommendations for the user and explain our proposed method for selection of the strategies using learning to rank algorithm. In our next chapter, we discuss the experiment we performed to evaluate our hypothesis, including an implementation of the overall system.

CHAPTER 4
EXPERIMENT

In this chapter, we discuss the experiment we performed in order to determine the effectiveness and accuracy of our proposed recommender system. We start with the overall experiment design and then discuss how data is collected and how the recommender system was implemented. Then we show the experiment results together with our analysis and discussion on the results.

4.1 Experiment design

We performed the experiment to validate the accuracy of our proposed recommender system. We devised an experiment the duration of which spanned a few months. We started with implementing the proposed recommender system. As we have defined the system in modules, we implemented different modules separately.

The first module that we implemented is the Data Crawler. As our recommender system is not directly connected to the Twitter Social graph repository, we downloaded the data and made it available to our system. We composed the list of seed users to be used for the download of data. Since our system generates final recommendation based on the historic followees-adding pattern of the user, the data crawler was scheduled to run for a couple of months in the interval of every two weeks so that we can use the last dataset as the current data and all other prior datasets as historic data.

We implemented the other modules of the system while we are collecting the data. Once we have sufficient number of datasets, we used the data from each of the datasets and calculated the scores using the different recommendation strategies mentioned in Chapter 3. Lastly we calculated our final recommendations using the various learning to rank algorithms and compared our results.

4.2 Dataset

We used Twitter data to generate recommendations in our experiment. In order to build the Twitter social network graph for our local repository, we downloaded the required data from Twitter repository using our Data Crawler. For each of the users for whom we want to generate recommendations, we required the following data:

1. List of followees
2. List of followers
3. List of followees of followees
4. List of followees of followers
5. List of followers of followees
6. List of followers of followers
7. Tweets and retweets of the user
8. Tweets and retweets of the followees
9. Tweets and retweets of the followers
10. Profile information such as name of the user, number of followees, number of followers, location, etc. of the user, user's followees and user's followers. We will not be using all the profile information but still decided to save it for any of our future work.

We composed our list of seeds by selecting random Twitter users who are followers of Ryerson University. There are a few Twitter profiles of Ryerson University such as @RyersonU, @RyersonNews, @TRSMRyersonU, @askmeRU, @ChangSchool, @theryersonian and @TEDxRyersonU. The table below shows the number of followers for each of the different accounts.

Table 1: Followers of Ryerson

Name	Screen Name	Number of followers
Ryerson University	@Ryerson	13587
Ryerson News	@RyersonNews	1906
Ted Rogers School	@TRSMRyersonU	815
Ryerson University	@askmeRU	1595
Chang School	@ChangSchool	563
The Ryersonian	@theryersonian	1274
TEDxRyersonU	@TEDxRyersonU	1519
	Total:	21259
	Distinct:	16184

Since there are some users who are members of more than one Ryerson account, we calculated the distinct number of users. Among these users, we eliminated users who are following more than 5000 followees, users who are following less than 10 followees and users who have less than 10 tweets. We randomly selected 500 users from the remaining followers of Ryerson University for our experiment, because the original number is too high for our available computing resources. However, by the time we actually completed our data collection, we found that 10 of the 500 users either deleted their profiles or made their Twitter account protected. It is not possible to collect data from protected accounts using Twitter API. So the effective number of users is 490 at the end. In this experiment, we will show results and analysis for these 490 users.

We used our data crawler to collect the required data. During every run of the crawler, we downloaded the complete list of information mentioned above for each of the users in the seed list. However, we defined a threshold for the number of tweets and retweets for the user, followees of the user and followers of the user. Since tweets and retweets are reflection of the user's interest and since interest changes over time, we considered the latest 800 tweets and retweets of the user, followees of the user and followers of the user.

Our recommender system uses historic data to compose the final recommendation. From April to July 2013, we ran our data crawler seven times each with interval of two weeks. Since Twitter has limitation on the amount of API calls to download the data, we used 5 computers and 7 Twitter accounts for each machine, a total of 35 crawler instances to download the data. Once the data is downloaded, it is saved in the SQL Database.

4.3 Implementation

Each of the modules of our system is implemented using Java Programming Language. We used Eclipse IDE for Java EE Developers as a compiler for the source code and Windows XP platform as the runtime execution environment for most of our programs.

Data Crawler is the first module we implemented so that we could collect data and implement other components at the same time. Our Data Crawler module uses a third party library Twitter4J to connect to the Twitter API 1.1. Twitter4J [36], written in java, is a reliable library to use for data collection. Twitter API is Twitter's interface for the developers and researchers to get access to the Twitter's social network data. It provides various functions to get data such as the list of followees of a given user, the list of tweets of a user, etc. Each function-

call to this API returns the data in JSON (JavaScript Object Notation) format, which can be parsed using the Twitter4J library functions and saved in the File Structure by our Data Crawler.

We used both File Storage and SQL Database for storage. We used Windows File Directory for File Storage. All social network data that our crawler downloaded was first stored in text files located in our File Storage System. Upon completion of the download, the data is cleaned and transformed into SQL queries for insertion into the Database by the Data Processor module. Cleaning includes removing any extra spaces and special characters that can cause problem to SQL insertion. We warehoused the prepared data and result of our score calculation in MySQL Server 5.5 Database. MySQL is a robust Relational Database Management System that is light-weight, easy to install and easy to use. We used MySQL Community Edition that is freely available under the GPL License. Using MySQL Workbench 5.2 CE, we were able to view the data interactively. We created different tables in the database to store the various social network data. Among the list of tables are: tblUsers, tblFollowees, tblFollowers, tblTweets, tblMentions, tblRetweets.

The Score Generator module is a collection of smaller modules each of which is responsible to calculate scores using a recommendation strategy. Among the sub-modules are Retweet PageRank Calculator, Similarity Score Calculator, Most Mention Score Calculator, Followees-of-Followees Score Calculator, Followee-of-Follower Score Calculator, Follower-of-Followee Score Calculator and Follower-of-Follower Score Calculator. Each of the sub-modules is separately implemented. They use input data from the SQL Database, calculate the score using specific recommendation strategies and write the calculated score to the Database.

We used JUNG [37] Java library (v2.0.1) to implement Retweet PageRank Calculator. Retweet PageRank Calculator is a resource intensive calculation as it involves ranking users for

the entire social network graph using retweet relationships as discussed in the earlier chapter. Similarity Score Calculator is another resource intensive calculation as it compares the contents of the user to the contents of all other users in the network. In order to calculate these scores within reasonable timeframe, we utilized the resources available in SHARCNET. SHARCNET [38] is a network of high-performance computers from different Canadian Universities that is available for researchers. It is a sub-division of Compute Canada and provides high-end computing resources such as computers, storage, networking and visualization to solve computationally intensive complex problems. We performed execution of Retweet PageRank Calculator and Similarity Score Calculator using resources from SHARCNET.

For Similarity Score Calculator, we require to compare contents of user's tweets to the contents of all other users in the network. One of the efficient ways to do this is to create documents for each user and index the documents based on its contents. We used Lucene v4.2.1 external library to aid in our calculation. Apache Lucene Core [39] is a Java-based high-performance library that supports text indexing, search and analyzing. Different analyzers are available in the package. The one we used provides option to include stop words list. We used a list of common words as our stop words list. Lucene provides a feature to calculate document similarity called MoreLikeThis. MoreLikeThis calculates similarity between documents using Vector Space Model (VSM) and Cosine Similarity and outputs the ranking scores. We used Lucene library to index the documents containing tweets of the users from the social network, followed by the usage of MoreLikeThis feature to generate the ranking scores.

The last two modules of our system, Rank Input Generator and Ranking Component are the most important modules as they generate the final recommendations. Rank Input Generator creates the input files for the learning to rank algorithms and the Ranking Component consists of

the learning to rank algorithms. In our experiment, we devised three different ways to calculate the final recommendations. The approaches are:

1. Using one strategy at a time.
2. Using linear combination of all the different strategies with equal weights.
3. Using historic data and learning to rank algorithm.

Among the three approaches, we used the first two as our baselines in later experiment, and the last one is our proposed algorithm. We used seven different learning to rank algorithm and evaluated their results to identify which ones are the best for the type of the data we use to generate recommendations. The learning to rank algorithms that we used are: MART (Multiple Additive Regression Trees), RankNet, RankBoost, AdaRank, Coordinate Ascent, ListNet and Random Forests. We used RankLib v2.1 package for learning to rank algorithms. RankLib [40] is a library of learning to rank algorithms. All the learning-to-rank algorithms we are using for our research are available in RankLib library. Not only do we use RankLib package for learning to rank algorithms but also for normalizing all scores from different strategies. Our Rank Input Generator was used to generate training, validation and test data for the learning to rank algorithms. Historic data was used to generate training and validation data whereas the current data is used to generate the test data. The last dataset is used for calculating the relevancy factor of the current data. The second last dataset is used as the supplier for the current data, whereas all datasets prior to this second last dataset is used as the supplier for historic data. Details about how the historic data is divided into training and validation data are described in the next sub-section.

The input files to the learning to algorithms follow the SVM-Rank format. A sample format is shown below:

```
<relevancy> qid:<datasetId> <featureId_1>:<score> … <featureId_N>:<score>
0 qid:3 1:64.0 2:54.0 3:31.0 4:25.0 5:356.0 6:0.01954206542 7:0.020361286
1 qid:8 1:13.0 2:1.0 3:3.0 4:0.0 5:781.0 6:0.02467538727 7:0.0
```

In the above sample, the first line shows the column names. Second and third lines are examples of the input data for the learning to rank algorithms. The first column represents the relevancy score. A score of 1 represents that this recommended candidate was added by the user as a followee, whereas a score of 0 represents that the recommended candidate was not added by the user as a followee. The column "qid" represents the dataset identification number or in RankLib term query Id. In the example above qid:8 is a snapshot taken later than qid:3. Rest of the data for each line represents the set of feature ids and their corresponding scores. Since we are using seven recommendation strategies, we have 7 sets of feature ids and scores. All the scores of the features are calculated using the dataset represented by qid and the relevancy score is obtained from the dataset right after the one represented by the qid. For the first line of data in the above example, all the features scores are calculated using dataset number 3 and the relevancy score is obtained from dataset number 4. The aim is to find what conditions (features and scores) are responsible for the acceptance of the recommended candidate (as shown by the relevancy score).

Based on the resources available, we decided to choose only a subset of the seeds for analysis. From the 490 seeds, we selected 50 most active users for whom we generate recommendations in our experiment. We generated recommendations using the three different approaches and using all the available learning to rank algorithms in RankLib. Using proper evaluation metrics, which will be discussed in the next section, we compared the result and proved that our proposed recommender system performs the best.

4.4 Results and Analysis

We generated recommendations for our experiment using the three approaches. We collected 7 snapshots of Twitter data at the interval of two weeks. Table 2 shows how these 7 seven datasets were used to generate strategy scores that were used for the learning to rank algorithms.

Table 2: Generate input for learning to rank algorithms

Dataset	Ranking Input	qid	Input Type
Apr-26	Scores	3	Training
May-10	Relevancy factor		
	Scores	4	Training
May-24	Relevancy factor		
	Scores	5	Training
Jun-07	Relevancy factor		
	Scores	6	Training
Jun-21	Relevancy factor		
	Scores	7	Validation
Jul-05	Relevancy factor		
	Scores	8	Test
Jul-19	Relevancy factor		

As shown in the above Table, our historic data consists of training and validation data. Test data is our current data.

The secret to an efficient recommender system lies in the ranking of the candidate recommendations. The recommender system not only generates the list of the candidates for recommendation but also ranks the recommendations. If the recommender system generates a list of 1000 recommendations for a user, the user is not shown all 1000 recommendations. Rather, the recommender system ranks the recommendation in descending order of importance and starts with showing the most important recommendation first. In most cases, the recommender system

identifies the first 10 or first 20 most important recommendations and presents those selected candidates to the user.

Our recommender system generates a list of candidates with associated scores from each of the recommendation strategies. Each candidate will have seven scores from seven different strategies. Each of the candidates will also have a relevancy score as explained in Chapter 3. The relevancy factor indicates if the recommended candidate is a successful recommendation later accepted by the user. The scores are the different attributes of the candidates. In order to explain how the ranking works, let us consider how to use just one strategy for recommendation. Let us assume that our candidates in the recommendation list have just one strategy scores and a relevancy factor. In order to generate the final recommendation, we need to sort this list in descending order of the strategy scores. The candidate with the highest score is at the top. This is now a list ranked using only one strategy. If the first candidate in the list has a relevancy factor of 1, it proves that our ranking order is effective in the recommendation process. If most of the candidates with relevancy factor of 1 are in the top positions of the list, it further proves the effectiveness of the ranking process. However, if most of the candidates with relevancy factor of 1 are at the bottom of the list, it shows that our ranking process is not effective.

In the presence of only one strategy, ranking is accomplished by simply sorting the list in descending order. However, ranking is not as easy as sorting one column in the case of multiple strategies. In that case, we first need to combine the scores from all strategies either linearly with equal weights or using some function. The resultant single score can then be sorted in descending order to generate a ranked list of recommendations. Learning to rank algorithm is used to derive this function which is often called the ranking model.

In this experiment, we used Mean Average Precision (MAP) as our evaluation metric. MAP value is used to determine the effectiveness of our ranked recommendation candidate list. For a given query, Average Precision is denoted by Equation (4.1).

$$AP(q) = \frac{\sum_{k=1}^{n} Precision(k) * RelevancyIndicator(k)}{m} \tag{4.1}$$

In Equation (4.1), *Precision(k)* represents the precision at position *k*, i.e. the number of relevant items in the first *k* items. *RelevancyIndicator(k)* represents the relevancy of the item at position *k*. If an item is relevant, *RelevancyIndicator* has a value of 1, otherwise 0. *n* is the total number of items associated with the query *q* and *m* is the total number of relevant items. In other words, *m* represents the total number of items whose *RelevancyIndicator* is 1.

Mean Average Precision is the mean of the Average Precision over all queries and is denoted by Equation (4.2).

$$MAP = \frac{\sum_{q=1}^{Q} AP(q)}{|Q|} \tag{4.2}$$

In Equation (4.2), Q is a set of queries and *AP=(q)* is the Average Precision for query *q*. In our case, the queries refer to the datasets, items refer to the candidates and features refer to the strategies. Our input file will have list of scores from strategies and a relevancy factor that identifies if the recommended candidate was later added by the user as a followee. For each of the strategies, we have a ranked list and MAP determines the effectiveness of the ranked list based on the relevancy factor. Since the ranked list is sorted in descending order, the items in the top of the list have higher scores than the ones at the bottom. If relevancy factor reveals that the items in the top are accepted more often than the items at the bottom then the MAP value will be high. If on the other hand, the relevancy factor indicates that the items at the bottom are accepted

more than the item in the top, then MAP value will be low. The higher the MAP value, the higher the effectiveness (or accuracy) of the ranked list. MAP value evaluates if the ranked list, or scores in the ranked list, is useful in determining the chance of the recommendation getting accepted.

For the first two of our approaches, we do not need training data as we are not trying to find the performance of the individual strategies. In the first approach, we used the scores from each of our strategies separately one at a time and got baseline MAP values. In this approach we tried to find how good each strategy works by itself in generating recommendations. In our second approach, we linearly combined the scores using equal weights and form another baseline MAP value. In our final and our own approach, we used the historic scores to build the model using learning to rank algorithms and later used the derived model to rank the recommendation list and calculate MAP values. Based on historic data, the model will know which strategies or combination of strategies works the best for the particular user. By comparing the baseline MAP values from first two approaches with the MAP values from our proposed approach, we will infer the effectiveness of our proposed approach.

In our first approach, we use the Test data as shown in Table 2. This consists of ranked lists of the scores from each of our recommendation strategies along with relevancy factor. We use seven strategies in our research. By calculating the MAP scores for each of the strategies, we aim to find if the strategies are useful in ranking the recommendations. In our second approach, we again use the Test data as shown in Table 2. However, instead of using the strategy-scores individually, we linearly combine the scores into one and use this resulting score to rank the recommendations. Using relevancy factor, we calculate the MAP score of the ranked recommendations.

In Figure 6, we show the MAP values of the recommendations for all 50 users using Strategy 1 only. The Y-axis shows the MAP values and the X-axis shows the list of the user IDs. As shown in the Figure, the MAP values varied among the users. For some users, this strategy was effective in ranking the recommendations while it was not so effective for some other users. Figures 6-12 show the MAP values using different strategies for the 50 selected users of our experiment. Figure 13 shows the MAP values using Linear Combination of all the strategies with equal weights for the 50 users.

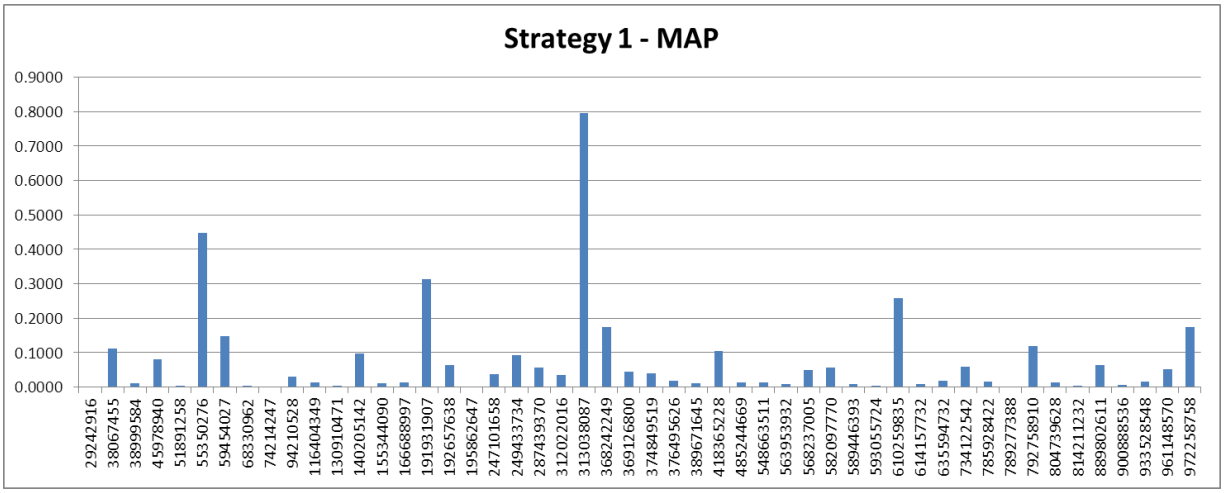

Figure 6: Comparison of Strategy 1 MAP for different users

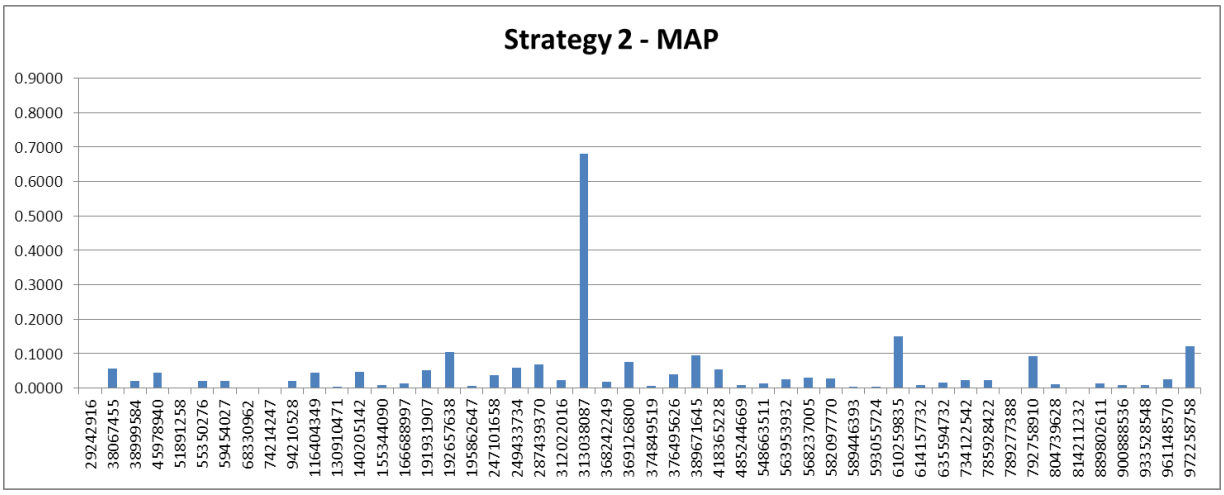

Figure 7: Comparison of Strategy 2 MAP for different users

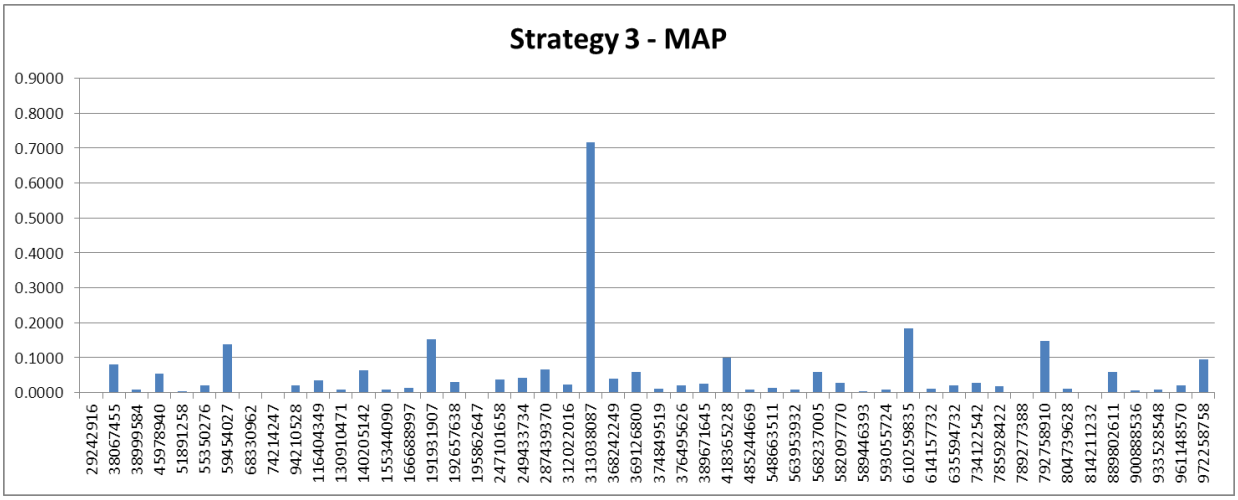

Figure 8: Comparison of Strategy 3 MAP for different users

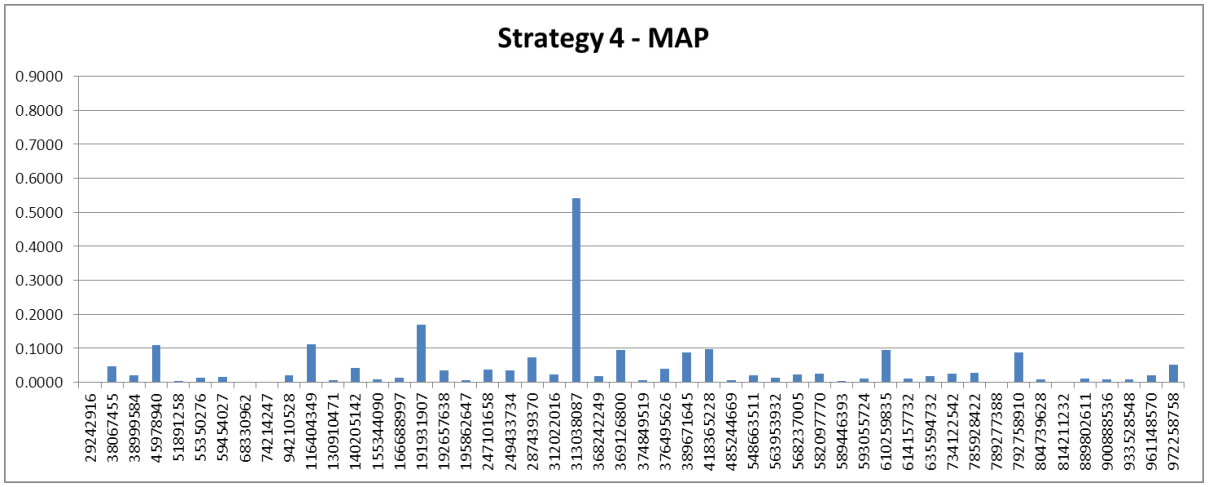

Figure 9: Comparison of Strategy 4 MAP for different users

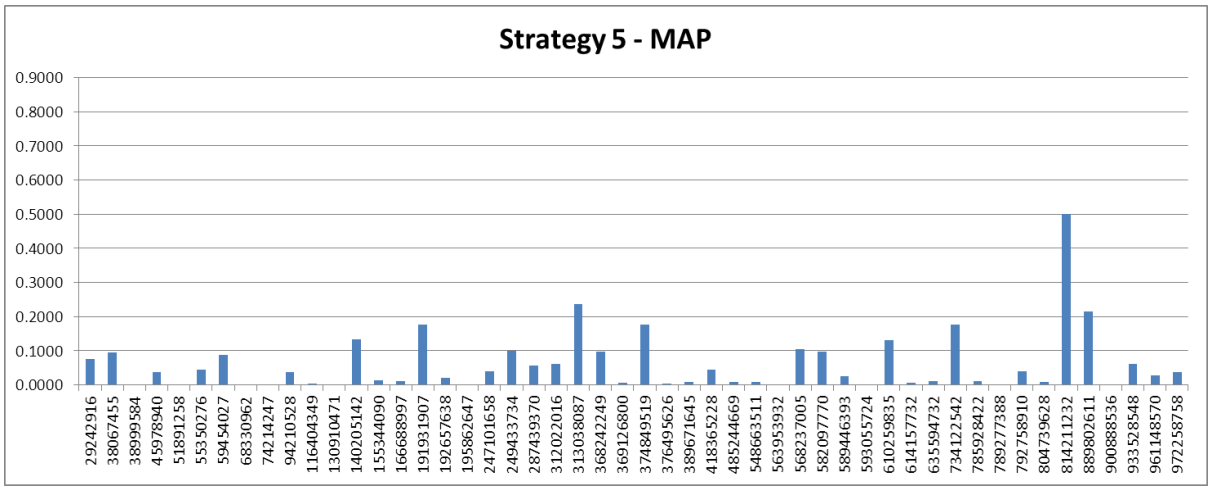

Figure 10: Comparison of Strategy 5 MAP for different users

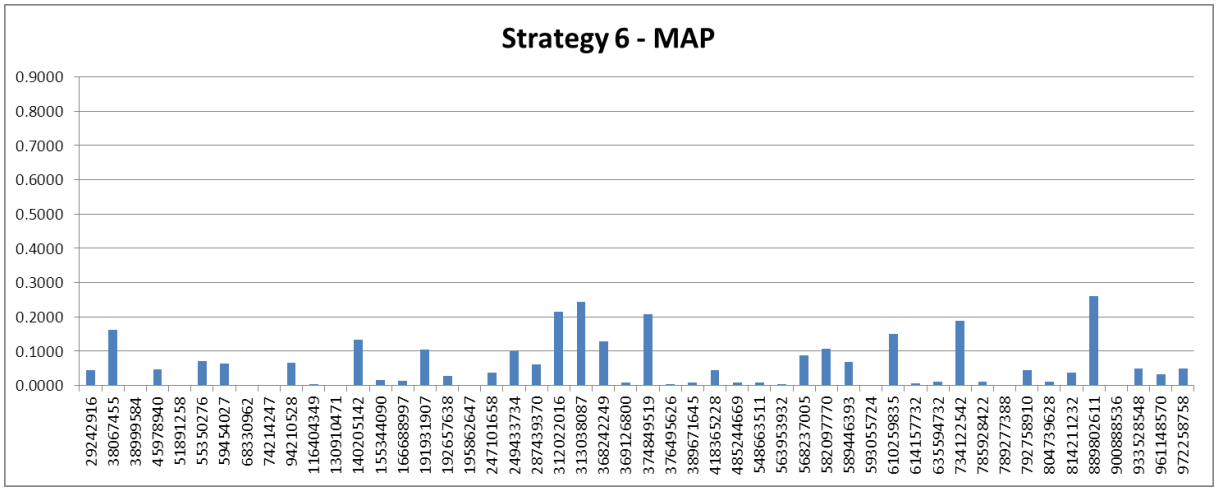

Figure 11: Comparison of Strategy 6 MAP for different users

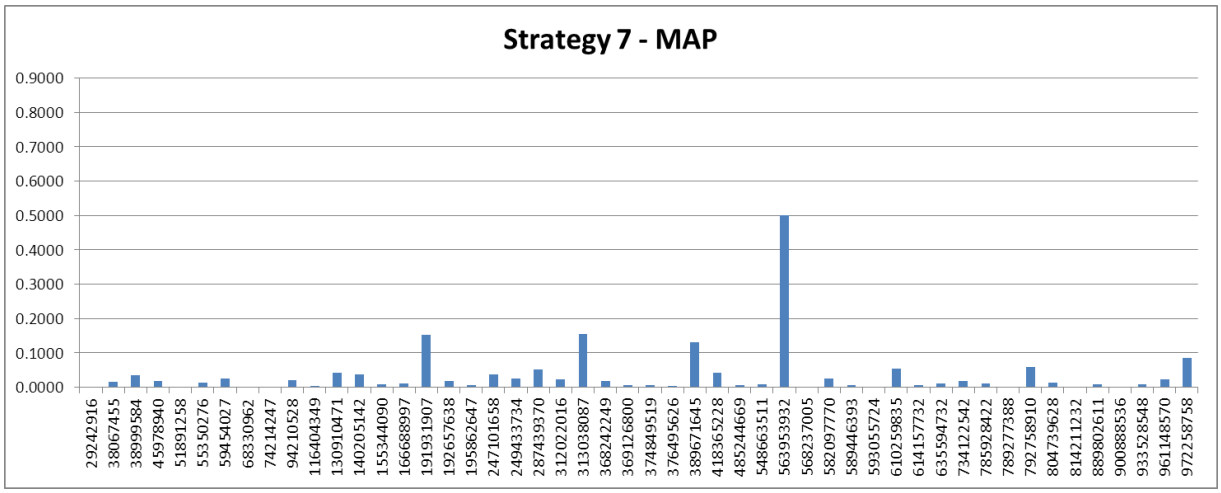

Figure 12: Comparison of Strategy 7 MAP for different users

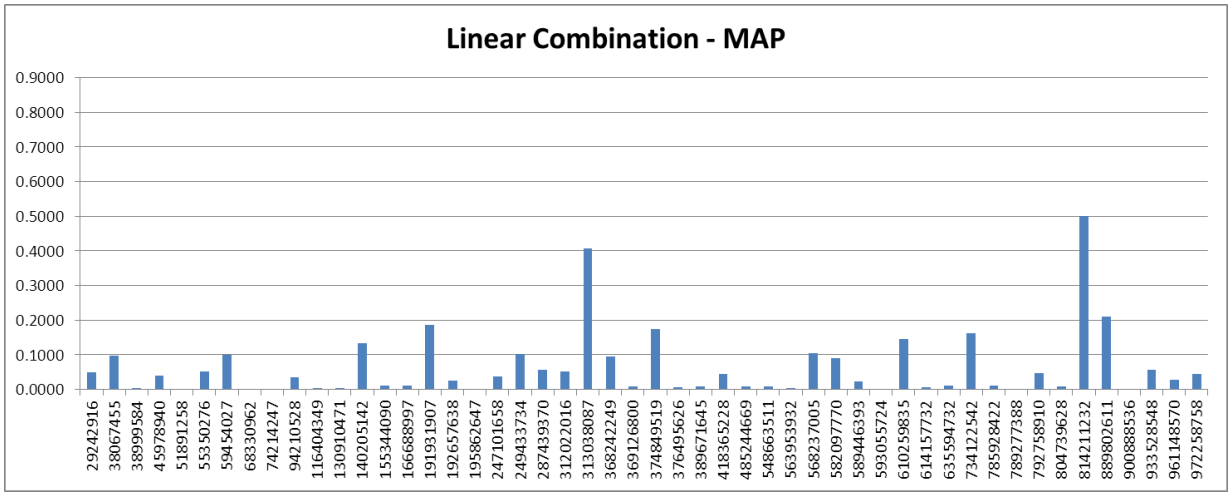

Figure 13: Comparison of Linear Combination MAP for different users

As seen in Figures 6-12, the strategies produced various results for different users. A single strategy may work for one user, but it may not work for the next user. The Linear Combination result as shown in Figure 13, did not either produce any better result. Figure 14 shows MAP of all strategies for a single user. For this particular user, Strategy 6 performed the best.

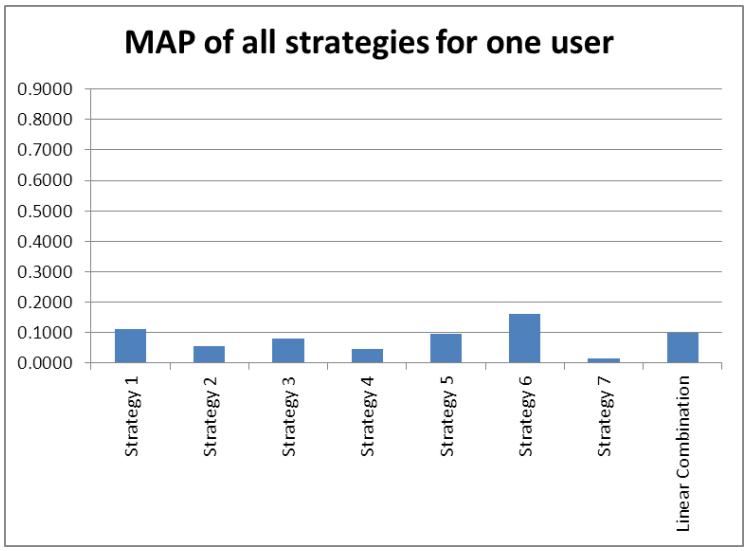

Figure 14: Comparison of MAP of different strategies for user 38067455

Figure 15 shows the MAP values of different strategies but for a user different than that of Figure 14. For this user, Strategy 1 performed the best.

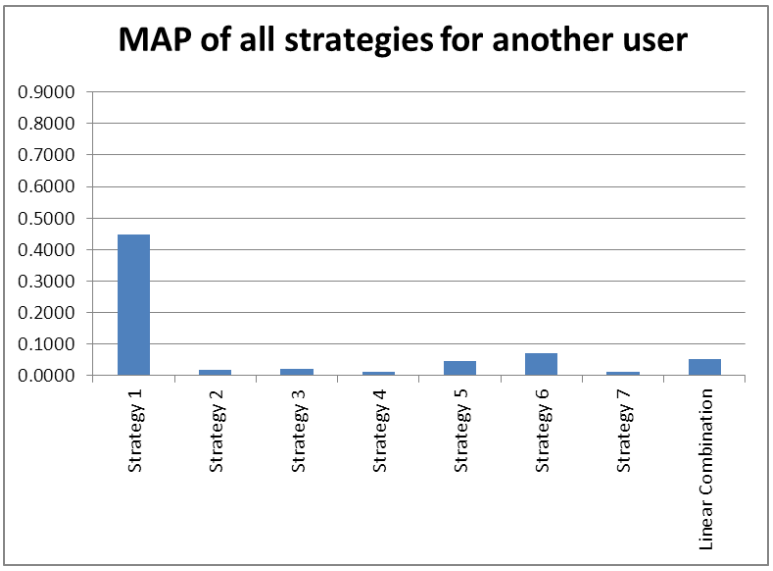

Figure 15: Comparison of MAP of different strategies for user 55350276

Figure 16 shows the comparison of different strategies and is a combination of the results shown in Figures 6-13; however, due to limitation of visualization, we show results for 18 users only. It is evident from the results displayed in Figures 16 that the MAP values are different for different strategies and they also vary from user to user. For some of the users, Strategy 1 performed the best, while for some others Strategy 2 performed the best, and so on. For some users, all strategies do not work well. The possible reason could be that they are using some strategies our system did not support, or there is a certain level of randomness in selecting users to follow.

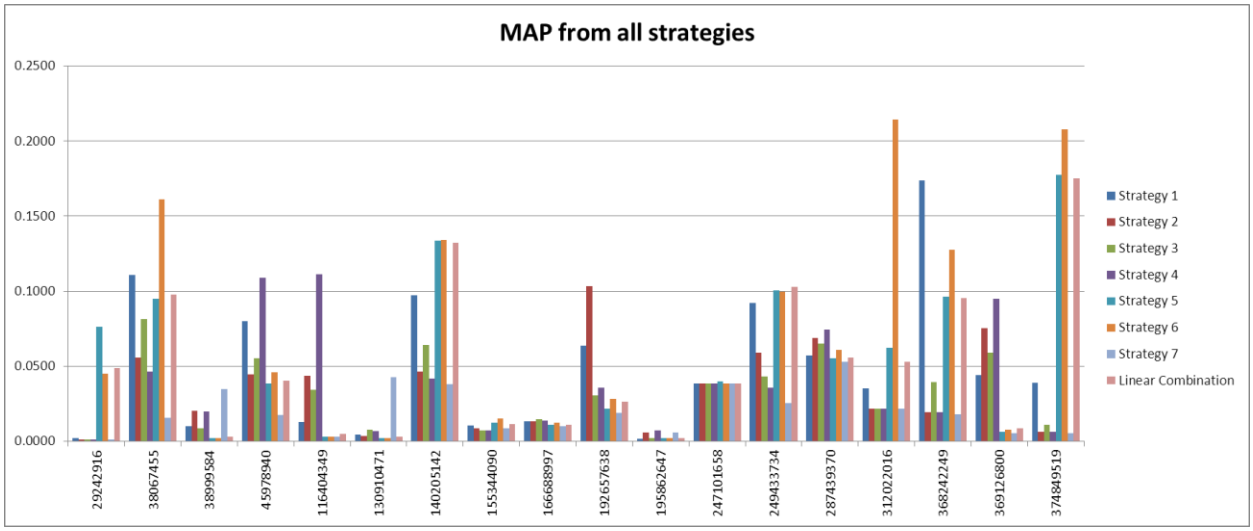

Figure 16: Comparison of MAP for different strategies and for different users

For further analysis, Table 3 shows the best count of strategies. Among the 50 user result analysis, we found that Strategy 1 performed best among all the strategies for 12 users, Strategy 2 performed best for 1 user and so on.

Table 3: **Best Result Count from all Approaches**

Strategy	Best count
Strategy 1	12
Strategy 2	1
Strategy 3	4
Strategy 4	9
Strategy 5	5
Strategy 6	10
Strategy 7	6
Linear Combination	3

We show the Average MAP scores from all three approaches in Table 4. For each of the approaches, we used the same 50 users to generate recommendations. In Approach 1, we calculated Average MAP for each of the individual strategies. In Approach 2, we calculated the

Average MAP for the linearly-combined-scores from all strategies. Lastly, in Approach 3, we used historic data as training and validation data to generate the ranking model and use the ranking model to generate recommendations. In this approach, the MAP is calculated for each of the learning to rank algorithms we used. These Average MAP values were helpful in determining which learning to rank algorithm(s) is best for our data. Besides showing the Average MAP, Table 4 also shows the Best MAP, Worst MAP and Standard Deviation for all the approaches. Standard Deviation is calculated as shown in Equation (4.3).

$$Std\ Dev. = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.3}$$

Where x is the individual MAP, $\bar{x}$ is the mean MAP and n is the number of users considered which is 50 in this case.

Table 4: **Results from all Approaches**

		Average MAP	**Best MAP**	**Worst MAP**	**Standard Deviation**
Approach 1	Strategy 1 – Retweet PageRank	0.0745	0.7965	0.0012	0.1355
	Strategy 2 – Similarity	0.0446	0.6812	0.0006	0.0979
	Strategy 3 – Most Mention	0.0506	0.7175	0.0006	0.1056
	Strategy 4 – Followee of Followee	0.0432	0.5426	0.0006	0.0811
	Strategy 5 – Followee of Follower	0.0610	0.5000	0.0015	0.0882
	Strategy 6 – Follower of Followee	0.0592	0.2606	0.0012	0.0700
	Strategy 7 – Follower of Follower	0.0353	0.5000	0.0005	0.0760
Approach 2	Linear Combination of all strategies	0.0647	0.5000	0.0014	0.0978
Approach 3	MART	0.7291	1.0000	0.0055	0.3588
	RankNet	0.3721	1.0000	0.0017	0.3465
	RankBoost	0.0541	0.6764	0.0006	0.1024
	AdaRank	0.0431	0.6719	0.0005	0.0982
	Coordinate Ascent	0.8448	1.0000	0.0004	0.2217
	ListNet	0.0874	0.6152	0.0005	0.1377
	Random Forests	0.8141	1.0000	0.0157	0.2658

In Figure 17, we show a comparison of the Average MAP values for all approaches. The first seven values are from approach one where we used individual strategies to recommend and use their individual MAP values to evaluate. The eighth value from the left corresponds to our approach two. The last seven results on the right are from approach three. These bars show the performance of each of the recommendation algorithms and learning to rank methods.

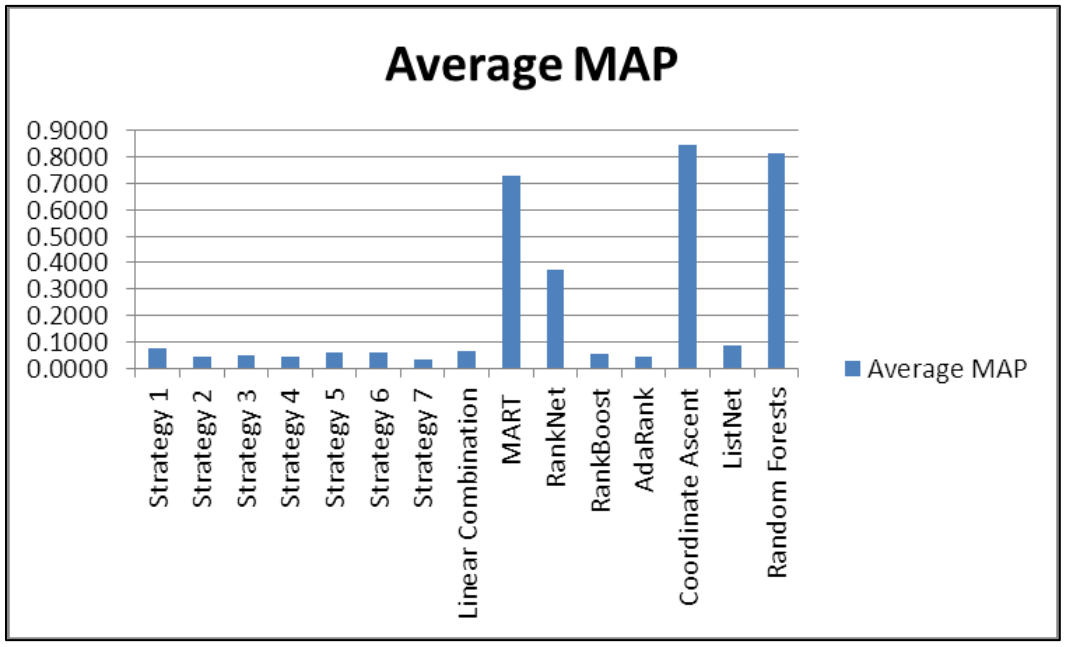

Figure 17: Average MAP of all approaches

Upon analyzing the results in Figure 17, we realize that Strategy 1 (Retweet-PageRank) performed the best among all the recommendation strategies we used. Not all the strategies performed the same. However, there is not much improvement in performance when we linearly combined the strategies. The performance obtained using the learning to rank methods is significantly higher than those of using the strategies individually or linearly combining them. Not all learning to rank method performed high. Learning to rank methods such as RankBoost,

AdaRank and ListNet performed similar to the individual strategies; whereas methods such as MART (Multiple Additive Regression Trees), RankNet, Coordinate Ascent and Random Forest performed better. As seen from Figure 17, the results from using these learning to rank methods are 5 to 8 times better. Learning to rank algorithms played an important role in personalizing the recommender system for every user.

4.5 Summary

In this chapter we explained the dataset we used in our experiment and the experiment design and provided details of the implementation. By analyzing and evaluating our results, we proved that our proposed approach performs better than the conventional approach. By comparing the results obtained using our approach with that of the conventional approach, we can see that even the lowest MAP score obtained from the learning to rank algorithm is better than the MAP scores using individual scores alone or the linear combination of all the strategies. Using many strategies is better than using one; however, using historic data to learn the importance of each strategy to the user and then applying the learning to rank and generate recommendations is even better.

CHAPTER 5
CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In this thesis, we proposed a personalized recommender system for whom to follow in Twitter. Our system uses the historic followee-adding pattern of the user in order to extract user's preference in adding new followee. The recommender system uses seven different strategies to compute scores that represent the importance of the recommended followees to the user.

We use strategies that find popular users, similar users, most mentioned users, mutual followees of followees, mutual followees of followers, mutual followers of followees and mutual followers of followers. Candidate recommendation list is ranked using scores for each of the strategies. However, since the strategies calculate scores based on their individual algorithms, each one of them produces ranking in different order. We have shown using experiment results that the effectiveness of each strategy varies for different users. A strategy can be found to be effective for a user; however, the same strategy may not work for all users. If a user is interested to be friend with popular users, he may not be interested to use the strategy that recommends similar users. Some users may be interested in finding their close friends, while others may be interested in finding new followees who share their topic of discussion even if they do not know the new friends offline. A single strategy may not work for all users; similarly, a single linear combination of strategies will not work for all users. Some users may utilize just one strategy, while others may use a combination of all the strategies. In addition, different users use different combination of strategies. Although users may make wrong decisions of selecting followees, but if the amount of history data is big enough, the impact on the ranking model could be minimal.

In our research, we propose the recommender system that not only generates candidate list of proposed followees, but also ranks the candidate list based on user's previous followee-adding preference. The recommender system looks at the historic data of the user and tries to determine the strategy or combination of strategies that was used to add user's existing followees in the past. The system relies on learning to rank algorithm for building the model using historic data. The model can then be used to rank the final candidate list.

The effectiveness of our proposed system is illustrated by the results of our experiment. We conducted experiment using three different approaches. In the first baseline approach, we used scores of one strategy at a time to rank the candidate list. In the second baseline approach, we used a linear combination of all the strategy scores to rank the candidate list. In the final approach, which is our own proposed approach, we used learning to rank algorithm using historic data to rank the candidate list. By comparing results from our approach with the baseline results, we have shown that our system performs significantly better than the baseline approach.

The major contributions of our research are:

- We designed a recommender system that is personalized using the historic activity data of the user. Each user will have his/her own personalized model that can be used to rank the recommendation candidates.
- We proposed a novel approach to find the popular user using PageRank of retweet links.
- We designed and implemented a system that consists of a data crawler, data processor and recommender system.
- We collected social network data for 3 months. This dataset consists of data of the same users but from different timelines. It is used for our experiment and could be made available to other users who are doing similar research.

5.2 Future Work

We would like to continue working on our recommender system to increase its efficiency in recommending followees. A few directions we may work on are:

First, we would like to increase the number of recommendation strategies. Currently we use seven recommendation strategies; however, our system is flexible to consider more strategies in the future. Beside the strategies we are currently using, we like to introduce new recommendation strategies that can improve the ranking of the candidate recommendations.

Second, we would like to find other learning to rank algorithms that can be used for faster ranking. Configuring the current learning to rank algorithms to produce better results can also be an opportunity to increase the performance.

Third, on top of the followee-adding pattern of the user, we may also consider the followee-deletion pattern, which may further improve the ranking accuracy. As we mentioned earlier, in our current system, if a user made a wrong decision on following another user, it might affect the accuracy of the learned model, especially when the amount of history data is small. However, if we also consider what followees a user deleted, the effect of adding wrong followees on the ranking model can be reduced, with the assumption that the user could find his/her mistakes later.

REFERENCES

[1] C.C. Aggarwal, "An Introduction to Social Network Data Analytics" in "Social Network Data Analytics", Kluwer Academic Publishers/Springer, pp. 1-14, 2011.

[2] A. Java, X. Song, T. Finin, and B. Tseng, “Why we twitter: understanding microblogging usage and communities”, in Proceedings of the 9th WebKDD and 1st SNA-KDD 2007 workshop on Web mining and social network analysis, pp. 56-65, 2007.

[3] J. Hannon, M. Bennett and B. Smyth, “Recommending Twitter Users to Follow Using Content and Collaborative Filtering Approaches”, in Proceedings of the fourth ACM conference on Recommender systems, pp. 199-206, 2010.

[4] H. Kwak, C. Lee, H. Park, and S. Moon, “What is twitter, a social network or a news media?”, in Proceedings of the 19th international conference on World wide web, pp. 591-600, 2010.

[5] J. Chen, W. Geyer, C. Dugan, M. Muller and I. Guy, "Make new friends, but keep the old: recommending people on social networking sites”, in Proceedings of the 27th international conference on Human factors in computing systems, pp. 201-210, 2009.

[6] P. Gupta, A. Goel, J. Lin, A. Sharma, D. Wang and R. Zadeh, “WTF: The Who to Follow Service at Twitter”, in Proceedings of the 22nd international conference on World Wide Web, pp. 505-514, 2013.

[7] W.H. Hsu, A.L. King, M.S.R. Pardesi, T. Pydimarri and T. Weninger, “Collaborative and Structural Recommendation of Friends using Weblog-based Social Network Analysis” in AAAI Spring Symposia 2006 on Computational Approaches to Analyzing Weblogs, pp. 55-60, 2006.

[8] S. Brin and L. Page, "The anatomy of a large-scale hypertextual Web search engine", in Proceedings of the WWW7 Proceedings of the seventh international conference on World Wide Web, pp. 107-117, 1998.

[9] C.D. Manning, P. Raghavan and H. Schütze, “Link Analysis” in “An Introduction to Information Retrieval”, Cambridge University Press, Cambridge, England, pp. 464-470, 2009.

[10] C.D. Manning, P. Raghavan and H. Schütze, “Scoring, term weighting and the vector space model” in “An Introduction to Information Retrieval”, Cambridge University Press, Cambridge, England, pp. 109-133, 2009.

[11] F. Ricci, L. Rokach and B. Shapira, “Introduction to Recommender Systems Handbook” in “Recommender Systems Handbook”, Springer, pp. 1-35, 2011.

[12] J.B. Schafer, D. Frankowski, J. Herlocker and S. Sen, “Collaborative filtering recommender systems” in The Adaptive Web, pp. 291–324, Springer Berlin/Heidelberg, 2007.

[13] K. Shen, J. Wu, Y. Zhang, Y. Han, X. Yang, L. Song and X. Gu, “Reorder User’s Tweets” in ACM Transactions on Intelligent Systems and Technology, Vol. 4, No. 1, Article 6, Jan. 2013.

[14] T.Y. Liu, "Learning to Rank for Information Retrieval" in Journal of Foundations and Trends in Information Retrieval, Vol. 3, Issue 3, pp. 225-331, March 2009.

[15] J.H. Friedman, “Greedy function approximation: A gradient boosting machine”, in The Annals of Statistics, Vol. 29, No. 5, pp. 1189-1232, 2001.

[16] C.P. Lee and C.J. Lin, “Large-scale linear RankSVM,” Tech. Rep., 2013.

[17] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender, "Learning to Rank using Gradient Descent" in Proceedings of the 22nd international conference on Machine learning, pp. 89 - 96, 2005.

[18] Y. Freund, R. Iyer, R.E. Schapire and Y. Singer, "An Efficient Boosting Algorithm for Combining Preferences" in Journal of Machine Learning Research, Vol. 4, pp. 933-969, 2003.

[19] J. Xu and H. Li, "AdaRank: A Boosting Algorithm for Information Retrieval" in Proceeding of the 30th annual international ACM SIGIR conference on Research and development in information retrieval, pp. 391-398, 2007.

[20] D. Metzler and W.B. Croft, "Linear feature-based models for information retrieval" in Journal of Information Retrieval, Vol. 10 Issue 3, pp. 257-274, June 2007.

[21] Z. Cao, T. Qin, T.Y. Lin, M.F. Tsai and H. Li, "Learning to Rank: from Pairwise Approach to Listwise Learning to Rank using Gradient Descent Approach" in Proceedings of the 24th international conference on Machine learning, pp. 129-136, 2007.

[22] L. Breiman, "Random Forests" in Journal of Machine Learning, Vol. 45, Issue 1, pp. 5-32, 2001.

[23] J. Tang, S. Wu, B. Gao and Y. Wan, "Topic-level social network search", in Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 769-772, 2011.

[24] J. Chen, R. Nairn, L. Nelson, M. Bernstein and E.H. Chi, "Short and tweet: experiments on recommending content from information streams", in Proceedings of the 28th international conference on Human factors in computing systems, pp. 1185-1194, 2010.

[25] M. Pennacchiotti and A. Popescu, “Democrats, republicans and starbucks afficionados: user classification in twitter”, in Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 430-438, 2011.

[26] T. Lappas and D. Gunopulos, “Interactive recommendations in social endorsement networks”, in Proceedings of the fourth ACM conference on Recommender systems, pp. 127-134, 2010.

[27] J. Weng, E. Lim, J. Jiang and Q. He, "TwitterRank: Finding Topic-sensitive Influential Twitterers", in WSDM, pp. 261-270, 2011.

[28] P. Thonhauser, S. Softic and M. Ebner, "Thought Bubbles: a conceptual prototype for a Twitter based recommender system for research 2.0", in Proceedings of the 12th International Conference on Knowledge Management and Knowledge Technologies, Article No. 32, Sept. 2012.

[29] M. Tavakolifard, K.C. Almeroth and J.A. Gulla, "Does Social Contact Matter? Modelling the Hidden Web of Trust Underlying Twitter" in Proceedings of the 22nd international conference on World Wide Web companion, pp. 981-988, May 2013.

[30] M. AlMeshary and A. Abhari, "A recommendation system for Twitter users in the same neighborhood" in Proceedings of the 16th Communications & Networking Symposium, pp. 8-12, April 2013.

[31] A. Silva, S. Guimarães and M. Zaki, "ProfileRank: Finding Relevant Content and Influential Users based on Information Diffusion" in Proceedings of the 7th Workshop on Social Network Mining and Analysis, Article No. 2, pp. 10-19, Aug. 2013.

[32] H.B. Celebi and S. Uskudarli, "Content Based Microblogger Recommendation" in 2012 ASE/IEEE International Conference on Social Computing and 2012 ASE/IEEE International Conference on Privacy, Security, Risk and Trust, pp. 605-610, 2012.

[33] M.G. Armentano and A.A. Amandi, "A Topology-Based Approach for Followees Recommendation in Twitter", in 9th Workshop on Intelligent Techniques for Web Personalization and Recommender Systems, pp. 22-29, July 2011.

[34] R. Garcia and X. Amatriain, "Weighted Content Based Methods for Recommending Connections in Online Social Networks", in The 2nd ACM Workshop on Recommendation Systems and the Social Web, Barcelona, Spain, pp. 68-71, June 2010.

[35] S.A. Golder, A. Marwick, S. Yardi and D. Boyd, "A Structural Approach to Contact Recommendations in Online Social Networks", in Workshop on Search in Social Media, In conjunction with ACM SIGIR Conference on Informational Retrieval, pp. 1-4, 2009.

[36] "Twitter4J", http://twitter4j.org/en/, Last accessed: November 11, 2013.

[37] "JUNG – Java Universal Network/Graph Framework", http://jung.sourceforge.net/, Last accessed: November 19, 2013.

[38] "SHARCNET", https://www.sharcnet.ca/my/front, Last accessed: November 11, 2013.

[39] "Apache Lucene Core", http://lucene.apache.org/core/, Last accessed: November 11, 2013.

[40] "RankLib", http://people.cs.umass.edu/~vdang/ranklib.html, Last accessed: November 11, 2013.