GENERATING ARTIFICIAL DATA FOR SCALABILITY TEST OF A FOLLOWEE TWITTER RECOMMENDER SYSTEM

by

Jason Li

Bachelor of Science in Computer Science

Ryerson University, 2015

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, 2019

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.

GENERATING ARTIFICIAL DATA FOR SCALABILITY TEST OF A FOLLOWEE TWITTER RECOMMENDER SYSTEM

Jason Li

Master of Science in Computer Science

Ryerson University, 2019

Abstract

The current methods for the evaluation of the scalability of recommender systems measure the scalability of the whole application after deployment on a cloud by calculating the running times of the application when increasing the number of nodes. This method requires the complete development and implementation of a whole application. To be able to test the recommender system during the development phase, the major problem to test the scalability and accuracy is collecting real data (i.e. social data), which is a time-consuming task and sometimes it is not possible due to privacy concerns.

This thesis proposes measuring the scalability of Twitter recommender systems by simulating the software, which processes a large number of artificial tweets. A method is introduced and validated for producing artificial tweets to test a recommender system. This method of producing artificial tweets is based on using analytical modeling, tf-idf and bag-of-words model. A simulator is developed to test the scalability of a recommender system and underlying distributed environment.

Acknowledgements

I would like to express great gratitude to my supervisor, Dr. Abdolreza Abhari, for providing me an opportunity to work under his supervision. It is with his guidance and support that I am able to accomplish this goal of completing my thesis.

I would also like to thank fellow members of the DSMP (Distributed Systems and Multimedia Processing) lab for their support and guidance.

Finally, I would like to thank my family. My parents and my brother supported me greatly with their cooperation, patience, and encouragement to let me pursue my goal of completing my Masters.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ALS – Alternating Least Squares

API – Application Programming Interface

CCD – Cyclic Coordinate Descent

CDF – Cumulative Distribution Function

COV – Coefficient of Variation

df – Document Frequency

FIPA – Foundation for Intelligent Physical Agents

FN – False Negative

FP – False Positive

ID - Identifier

idf – Inverse Document Frequency

JADE – Java Agent Development Framework

MAE – Mean Absolute Error

NMF – Non-negative Matrix Factorization

RI – Rand Index

RMSE – Root Mean Square Error

ROC – Receiver Operator Characteristics

RT - Retweet

SGD – Stochastic Gradient Descent

tf – Term Frequency

tf-idf – Term Frequency Inverse Document Frequency

TN – True Negative

TP – True Positive

URL – Uniform Resource Locator

List of Algorithms

1 Introduction

1.1. Motivation

Twitter is an online social network service where users can post information in short messages, known as tweets, up to 280 characters. In these short messages, URL links and images can be posted as well to relate to the topic of the message. In the social network aspect of Twitter, a user can follow another user to be considered a follower while the other user would be considered a followee [1]. A graph of the network can be created based on the "following" relationship among the users interpreted as the edges of the graph that can be bi-directional. User A following User B does not necessarily mean User B follows User A back. The information is propagated through the network because there are some users that are information sources and some users that are information seekers [1], [2]. The information seekers may follow multiple followees that are information sources and the information seeker may have followee numbers greater than the follower numbers, in contrast to users that are information sources who may have follower numbers greater than their followee numbers.

Twitter will continue to grow as data continue to grow worldwide. A user becomes overwhelmed by the excess of information as the abundance of data is prevalent. To help users in comprehending the massive influx of information, recommender systems are designed to provide items of interest to users. In terms of Twitter, a recommendation can consist of a list of users that a user might be interested in following. However, testing the performance (i.e. scalability and accuracy) of such recommender systems is not an easy task.

Simulation is a way to test in the development stage when developing a production-level recommender system is not feasible. A developed simulator can simulate all methods for recommendation, for example, using the similarity of features between Twitter users' tweets or grouping users into communities and then recommending follower/followee by finding the neighbourhood within a community. In the latter example, cluster algorithms such as k-means can be used to group users.

The aim of this thesis is to address the problem of measuring scalability of such a recommender system for the social networking site of Twitter. To be able to test scalability, collecting a large amount of real data (i.e. tweets) is required, which is a difficult task. The reason

for this is two-fold: first, testing of scalability should be combined with testing of accuracy. For the scalability test, a large amount of data is needed, which is time-consuming because of the limitation set by the Twitter API in collecting data. Testing accuracy of a recommender system, the same as the one used in this thesis with the focus of producing follower/followee recommendations, requires collecting the user network graph in different times of the months, which is also very time-consuming. Secondly, collecting real data is also subject to getting permissions and privacy concerns. Even if permission is given, people have concerns about how their tweets are being used.

1.2. Problem Statement

The problem statement of this thesis is whether it is possible to generate artificial text data for Twitter that can be used to simulate real text data for testing scalability of a recommender system. Testing the scalability of recommender systems is an important issue and requires a large amount of data. The common way to address scalability is to use recommender systems in parallel on distributed systems such as cluster or cloud computing. The growing amount of data is not very feasible to just run a recommender system in a central server. A recommender system should be scalable to accommodate the large amount of data. As discussed in the previous section, collecting real data to be able to test both scalability and accuracy of recommender systems in social networking, such as Twitter, is a difficult task. Thus, the question is how to generate tweets based on a stochastic process.

Another problem is to be able to measure the timing performance and accuracy of the above recommendation system for the scalability test. To be able to do that, a simple case of recommendation, which is recommending followee/followers, is considered in this work. For simplicity, k in k-means clustering is set to a fixed cut-off value of 3, where the initial centroid of the first cluster is always fixed (which is a user for whom a list of followers will be recommended) and the two other initial centroids are selected randomly. The accuracy is achieved by measuring against the real followee/follower data. For example, by knowing the real followers of a user for whom the recommendation list will be generated, the accuracy of the system can be estimated for different scenarios. The hypothesis is the performance of time complexity for generating clusters of this scenario (recommending followers to the initial centroid of the first cluster) is similar to time complexity performance of a followee/follower recommendation system if someone builds

for Twitter by using k-means clustering for recommendation based on tweet similarity when considering the community of similar users.

1.3. Contributions

The proposed contribution is to provide a solution to this problem where it does not conflict with Twitter's policy, user's privacy, and provide a large amount of data to be used for this research.

The contributions provided by this thesis is as follows:

1. A novel method is proposed to generate artificial text data for Twitter
2. Simulation of a Twitter recommender system for suggestion of followers/followees
3. Development of a multi-agent system to simulate a distributed environment for different configurations of parallel running of a recommendation algorithm to test both scalability and accuracy of a recommendation algorithm.

Publication related to this work can be found in

- J. Li and A. Abhari, "Generating stochastic data to simulate a twitter user," in *Proceedings of the 20th Communications & Networking Symposium*, Virginia Beach, VA, USA, 2017, pp. 102–112.

1.4. Thesis Outline

The thesis is outlined as described in the following. Chapter 2 contains the background and literature review. Chapter 3 presents the methodology and performance metrics of the experiments. Chapter 4 contains the implementation and results of the experiments using the methodology in the previous chapter. Chapter 5 contains the conclusion, the contributions, and future works that can be developed from this research.

2 Background Information and Related Works

2.1. Background Information

In this section, the terms and models related to this work are explained in order to understand basic components, methodology and simulation used in this thesis.

2.1.1. Recommender Systems

Recommender systems are used to recommend items that are useful to users. These systems are needed because of the information overload problem or big data [3]. In general, recommender systems make use of three sets of elements. These elements are items, users, and transactions [4].

Items are domain relevant items in the recommender system. They can be denoted as a set $T = \{t_1, t_2, .., t_m\}$ [4]. The items can be represented based on its features. An example would be for movies, its features can be the actors in the movie and the genre of the movie.

Elements in the user set are users who have browsed the items or given feedback to provide ratings to the items. Users can be denoted as a set $U = \{u_1, u_2, \ldots, u_k\}$[4]. For collaborative filtering, users are represented in a model that has a list of ratings for the items in the item set.

Transactions are recorded interactions between the user and the recommender systems. The interactions are stored to provide information to the recommender system for its recommendation algorithm. The rating by the user can be explicit where the recommender system will explicitly ask the user for a rating to an item [5]. The explicit ratings can be numerical, binary, ordinal, or unary. Implicit ratings are where the recommender system looks at the behaviour of the user such as whether an action by a user was conducted to determine interest indicators [6].

A user-item matrix is a structure that holds the explicit or implicit ratings of a user to an item. The user-item matrix X is K×M where K is the number of users and M is the number of items. Each cell in the matrix is $x_{k.m}$ which holds a rating $r = \{1,\ldots. |r|\}$ that was given by a user to an item [4]. If the rating is unknown, the cell holds the value 0.

The recommender system is trying to solve the recommendation problem $f: U \times T \rightarrow R$ [4], [7]. U is the set of users in the system, T is the set of items in the system, "R is an ordered set of utility functions such that $f(u_k,t_m)$ computes the usefulness of item *m* to user *k*" [4]. There are several techniques used to solve this problem. The techniques that are more common are content-

based and collaborative filtering. Content-based recommends items based on the features/traits of items the user favoured in the past [8]. Collaborative filtering identifies similar users or items to a target user by a similarity function for recommendation to the target user [4]. The similarity function is to determine whether the ratings given by one user is similar to another user. Collaborative filtering can be memory or model-based [9]. Memory-based requires all the items and ratings of the users to be stored in memory. Model-based tries to learn a model based on the items and ratings to predict which group of users a user is most similar to [4].

Collaborative filtering consists of two tasks, rating prediction and recommendation. The rating predicted is a numerical value for an unseen item to the target user. The recommendation task gives the Top-N items that the target user is likely to give the highest rating. Model-based collaborative filtering is pre-computed using machine learning or rule-based approaches and considered efficient. Memory-based collaborative filtering does not do any computation until a recommendation is requested but has the advantage of obtaining the most recent transactions [4].

To find similarity between users or items, there are several similarity metrics. The metrics are cosine similarity, Pearson correlation, Spearman's correlation coefficient, adjusted cosine similarity, mean squared difference, Jaccard coefficient, and conditional probability-based similarity [4]. Cosine similarity takes vectors of ratings of features from two different users and calculate the angle between the vectors.

$$CV(u,v) = \cos(X_u, X_v) = \frac{\sum_{i \in I_{uv}} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I_u} r^2{}_{ui}} \sqrt{\sum_{i \in I_v} r^2{}_{vi}}} \quad (1)$$

Vectors that are perfectly similar results in a value of 1 since $\cos 0$ is 1.

The accuracy of collaborative filtering can be measured by predictive accuracy metrics, decision-support accuracy metrics, and rank accuracy metrics. Predictive accuracy metrics evaluates the true rating to the predictive rating of the recommender system. MAE (mean absolute error), or RMSE (root mean square error) are typically used for predictive accuracy metric. Decision-support is the evaluation of the acceptance rate provided by the user. Reversal rate, ROC (receiver operator characteristics) curve, precision and recall are typically used for decision-support metrics. Rank accuracy metrics measure the suggestions of a recommendation algorithm given a user's preferences [4].

A well-known problem known as the cold start problem plagues collaborative filtering. The cold start problem is when there is no previous data for a new user or item to a recommender system. Since there is no previous data (i.e. ratings), a new user cannot be easily compared to other users. Liu et al. in [10] tries to combat this problem using seeds, a known group of users that represents part of the population, and matrix factorization, which is a popular method for recommender systems. Matrix factorization is explored in different works in [11]–[15]. Yu and Qiu in [16] used matrix factorization to find followee recommendations in a microblog like Twitter.

2.1.2. Vector Space Model

For this thesis, a document is referred to the tweet of a user of Twitter. A tweet can contain different information like the location of where the tweet was posted, the date and time the tweet was posted, a unique identifier for the tweet, who posted the tweet, the text message, and much more. The documents for this study are focused on the text messages provided in the tweets. The arrangement of the words is not considered since the study of the documents is following the bag-of-words model where the number of occurrences of the words matters but the order does not [17].

The documents, *d*, can be considered vectors that hold weights of the words used in the tweet in its components. The values of the weights are determined by *tf-idf* (term frequency inverse document frequency) which is the combined definitions of *tf* (term frequency) and *idf* (inverse document frequency) [17]. Term frequency is defined as the number of occurrences of a term, *t*, in a document where the words of a tweet are considered the terms. Formally, *tf* is defined as $tf_{t,d}$. The raw frequency of a term is not a good indicator of the weights in a document vector since it gives higher weights to words that are commonly seen for a topic. A technique to compensate for this is to use a document-level statistic. The document frequency, *df*, is defined as the number of documents that a term occurs in. The inverse of the document frequency is used to scale down the weights since the rarer terms that occur in lower number of documents would have a higher *idf* than a more frequent term that occurs in higher number of documents. The *idf* is formally defined as $idf_t = \log \frac{N}{df_t}$ where N is the number of documents in a corpus. The combined definition of *tf* and *idf* to assign the weight for term, *t*, in a document, *d*:

$$tf\text{-}idf_{t,d} = tf_{t,d} \times idf_t \quad (2)$$

The *tf-idf* will provide the following properties: the weight of term *t* will be the highest when it appears many times in a few documents and the lowest when the term appears in almost every document [17].

2.1.3. Analytic Modelling

Descriptive statistics is composed of statistics that summarizes the data in a study. Three common quantitative statistics are the mean, median, and mode. However, these three statistics are not enough to describe the nature of the data since it does not describe all the possible outcomes. A distribution of the data describes all the possible variations of data while each distribution can have its own mean, median, and mode. Probability distribution is the function that gives the probabilities of all the possible outcomes in the distribution of the data.

2.1.3.1. Weibull Distribution

Different distributions were examined to better describe the data in this study. Based on empirical measurement conducted in the experiments and in [18], Weibull distribution appeared to be a suitable model for generating artificial text tweets which is explained in this section. The Weibull distribution is a continuous distribution that is defined as follows [19]:

Density Function:

$$f(x) = \begin{cases} \alpha\beta^{-\alpha}x^{\alpha-1}e^{-\left(\frac{x}{\beta}\right)^{\alpha}} & if\ x > 0 \\ 0 & otherwise \end{cases} \quad (3)$$

Distribution Function:

$$F(x) = \begin{cases} 1 - e^{-\left(\frac{x}{\beta}\right)^{\alpha}} & if\ x > 0 \\ 0 & otherwise \end{cases} \quad (4)$$

Where α is the shape parameter and β is the scale parameter. A shape parameter determines the basic form of a distribution in its family of distribution. A scale parameter expands or compresses the distribution without changing the basic form [19]. The two parameters of a Weibull

distribution are non-negative. The common method for the estimation of these parameters is explained next.

2.1.3.2. Maximum likelihood estimate

To estimate the parameters of a given distribution, a method known as maximum likelihood estimate (MLE) can be used. MLE is maximizing the likelihood function which is the total or joint probability of the observed data points. Each data point is assumed to be generated independently from the others. The general step of the method is to find the joint probability density function of a given distribution for the observed data points. The joint probability density function describes how likely two events occur at the same time. To make the math easier, the log of the joint probability density function is used since the maximum of the log function is the maximum of the function [20]. Differentiating the log of the joint probability density function will obtain the derivative of the function. Setting the derivative of the function to 0 will find the maximum of the function. Finally, the equation can be re-arranged to solve for the desired parameters.

The MLE equations for a Weibull distribution are given as the two equations, equations 5 and 6 that must be satisfied in the following [19]:

$$\frac{\sum_{i=1}^{n} X_i^{\hat{\alpha}} \ln X_i}{\sum_{i=1}^{n} X_i^{\hat{\alpha}}} - \frac{1}{\hat{\alpha}} = \frac{\sum_{i=1}^{n} \ln X_i}{n} \tag{5}$$

$$\hat{\beta} = \left(\frac{\sum_{i=1}^{n} X_i^{\hat{\alpha}}}{n} \right)^{\frac{1}{\hat{\alpha}}} \tag{6}$$

To find $\hat{\alpha}$ in equation 5, it must be solved numerically by Newton's method or Newton-Raphson method. Newton's method is a method for finding an approximation of the roots of a function through successive iterations. If x_0 is the initial estimate of the approximation for a root, a better approximation for x_1 would be the following formula:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \tag{7}$$

The method is successively applied until the approximation for the root converges where convergence can occur when the approximation is to a certain number of decimal places like 8 for example. The general formula is the following:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{8}$$

For the Weibull distribution, an initial estimate, $\hat{\alpha}_0$, can be found in the following equation 9 as described in [19], [21], [22]:

$$\hat{\alpha}_0 = \left\{ \frac{\left(\frac{6}{\pi^2 \left[\sum_{i=1}^{n} (\ln X_i)^2 - \frac{(\sum_{i=1}^{n} \ln X_i)^2}{n} \right]} \right)}{n-1} \right\}^{-\frac{1}{2}} \tag{9}$$

This initial estimate is used when applying the Newton's method to estimate the shape, α, parameter in equation 5 which then can be used to find the scale, β, parameter in equation 6 for the Weibull distribution. The general recursive step for the iterations of Newton's method to solve for the shape parameter is in the following equation [19], [21], [22]:

$$\hat{\alpha}_{k+1} = \hat{\alpha}_k + \frac{A + \frac{1}{\hat{\alpha}_k} - \frac{C_k}{B_k}}{\frac{1}{\hat{\alpha}_k^2} + \frac{(B_k H_k - C_k^2)}{B_k^2}} \tag{10}$$

where

$$A = \frac{\sum_{i=1}^{n} \ln X_i}{n},\ B = \sum_{i=1}^{n} X_i^{\hat{\alpha}_k},\ C = \sum_{i=1}^{n} X_i^{\hat{\alpha}_k} \ln X_i,\ \text{and}\ H = \sum_{i=1}^{n} X_i^{\hat{\alpha}_k} (\ln X_i)^2$$

The shape parameter obtained from equation 9 will be used to solve for the scale parameter using equation 6. Then the shape and scale parameter of a Weibull distribution will be known. This method is used in the estimation of the parameters of Weibull distribution, which is used for modelling tf-idf in this thesis.

2.1.4. Multi-Agent Systems

Simulation of recommender systems, social network users, and distributed systems can be conducted by multi-agent systems. A multi-agent system is composed of autonomous multiple agents trying to achieve an objective [23], [24]. An individual agent is an entity whose state consists of mental components which are beliefs, capabilities, choices, and commitments [24].

Agents communicate with each other through messages and once receiving a message, it will alter its current mental state [24]. An agent will also execute its current commitments at its current time and beliefs may change after execution [24]. A multi-agent system also consists of multi-agent organizations and multi-agent environments [23]. The multi-agent organizations consist of organization rules that must be followed by the agents [23]. While the environment can be external to the system or related to the run-time infrastructure of the system [23]. The environment is where agents interact with each other or with sources like services that are external to agents [23]. The concepts of these terminology are high-level and can vary depending on the usage with no clear definition universally accepted in research [23].

2.1.5. K-means Clustering Algorithm

The recommendation algorithm used in the experiments uses k-means clustering. K-means requires the number of clusters to be known at the start of the algorithm. The initial centroids of the clusters are the initial seeds chosen by the algorithm. Data points in a dataset are assigned to clusters based on a distance measure between the data point and the centroids. Iteratively, the algorithm assigns data points to the centroids with the lowest distance. The centroids will also be updated after all the data points are assigned to a cluster or as the data points are assigned to the clusters. Clustering is considered complete when the data points are assigned to the same clusters as the previous iteration. Commonly, k-means uses the Euclidean distance in Euclidean space. For a vector space, the distance measure can use cosine similarity [25]. To retain the distance property of minimizing the distance, the distance measure can be modified to be 1-cosine similarity [26]. Since the most similar vectors are equal to 1, 1-cosine similarity will equal to 0 which is the minimum Euclidean distance in the positive number space. K-means clustering is the method for collaborative filtering in the recommender system simulator.

2.2. Related Works

Netlytic is a text and social network analyzer [27]. It is capable of capturing data from popular social media like Twitter, YouTube, Facebook, Instagram, or from a simple text file. It offers support for multiple datasets depending on the tier system that you are in. With the data, Netlytic can do text analysis with a keyword extractor and find words for assigned categories. The network analysis can discover ties in a name network and build a chain network. Finally, a final report can be made containing a map of geo-tagged posts, Top Ten Posters, Top Ten Most

Frequently Used Words, Top Ten Posters mentioned in messages, the Number of posts over time and the results from the text and network analysis. This is evidence of another work that uses social network data for analysis.

Yang, Liu, and Mo created an agent-based framework to simulate a Twitter financial community [28]. Data was collected through the Twitter API and different agents were used. The different agents had different message propagation rates. The work tries to simulate an event that happened previously in a 2013 Associated Press Hoax. With the use of agents, the results of the messages propagated by the agents were validated by the results of the actual event. Also, with the help of the agents, it was shown that removing critical nodes from the network reduces the spread of a malicious message. This work showed that a multi-agent system would be useful in simulating a Twitter network.

Hernández del Olmo and Gaudioso proposed a framework that extracted the essential features of a recommender system [29]. The proposed framework is meant to solve the issue where comparisons between recommender systems are often incompatibly evaluated. The different accuracy metrics, information retrieval measures, rank metrics, and other metrics were shown to explain that these metrics are not uniformly used by all recommender system evaluations [29]. Three classifications were made to describe the good behaviour of a recommender system, which are rating prediction, ranking prediction, and successful decision making capacity [29]. Accuracy and information retrieval measures fell into the third class, which is the focus for a general recommender system metric. The proposed framework is composed of two subsystems: interactive and non-interactive. The interactive subsystem guides a user, which means when and how each recommendation is shown to a user. The non-interactive subsystem filters which items are useful or interesting to be recommended to a user. A new metric is also introduced to help with the measure of the general framework.

Meyer, Fessant, Clérot, and Gaussier analyzes the performance of automatic recommender systems in the industrial context to propose a protocol to evaluate recommender systems [30]. This work defined the functionalities of a recommender system to consist of four main features: to help decide, to help compare, to help discover, and to help explore [30]. These features were maintained in the creation of the proposed evaluation protocol. The performance measures in the evaluation protocol were RMSE for decide, compatible rank index for compare, precision and average

measure of impact (AMI), introduced in this work, for discover, and a similarity matrix (Pearson correlation coefficient) for explore [30]. It is concluded that RMSE and the quality of recommendation is not correlated based on the performance of the segments of the different features.

Shani and Gunawardana provides a tutorial focused on evaluating recommender systems from an application-oriented view [31]. In other words, this work was made in mind for an application designer. The evaluation procedure is suggested to consist of goals and tasks. This work identifies three popular tasks as recommendation, utility optimization, and rating prediction [31]. The evaluation methods used are online testing, user studies, and offline testing. Online testing is a user interacting with the recommender system within the application. User studies is testing groups of test subjects that tries to represent a population of the users for the recommender system. Offline testing is using the previous interactions of real users for the recommender system to conduct a simulation. Performance metrics suitable to the three tasks are presented. The rating prediction task used RMSE as the metric. The recommendation task used precision and recall. Finally, the utility optimization task used R score, defined in this work [31].

Simulation-based techniques for evaluating the performance of recommender systems have also been addressed in the literature. Saga, Okamoto, Tsuji, and Matsumoto in [32] used a multi-agent system for simulating a collaborative filtering recommender system. Items of 5, 10, and 20 were recommended to the user agents but this was not used in the domain of Twitter. Capuruço and Capretz in [33] used Monte Carlo simulation to provide random ratings to items for a user to evaluate the accuracy of a recommender system. The study consisted of 27 users that provided 46 ratings to 25 movies. Another study used simulation with a synthesized dataset to simulate a recommender system using collaborative filtering for informal learning [34].

Although sanitization of data does exist [35], it does not provide a timely solution to gathering a large amount of data for testing both scalability and accuracy for Twitter data. Given the imposed limits to the Twitter API calls by rate limiting, a significant amount of time will be spent on gathering a large amount of data from Twitter which should be collected before sanitization of data.

Generating a large amount of artificial data has been previously proposed for database systems which deal with structured data such as tables, records, etc. SimSQL, for example, uses

machine learning Bayesian methods to generate stochastic data for database simulations [36]. The Synthetic Data Vault uses machine learning approaches to build generative models of relational databases as well [37]. In a previous paper [18], Weibull distribution was used to generate stochastic tweets which were used for simulating Twitter users. A simple similarity-based recommender system was simulated as well for a small amount of data in that work. In this thesis, a general form of the analytical model for social networking web sites is proposed.

Research has been done in modelling the tweets from Twitter by aggregating all of the tweets made by a particular user into a single document [38], [39]. However, these aggregated tweets were neither used to generate stochastic data nor were employed in any scalability test or simulation. With regards to simulating Twitter itself, the software package Hashkat was developed to simulate large social networks structure and analyze network topology growth as well as information flow within them using agent-based modeling [40]. The concentration of these works studies the topology of subnetworks, not generating artificial tweets.

Context-free grammar is one of the oldest methods available for generating a sentence. It generates a sentence given a set of recursive rules, known as the grammar, consisting of symbols for terminals, non-terminals, and a start symbol with a given vocabulary [41]. To be able to produce a sentence, it would require manually hard-coding of the symbols of every word used in the vocabulary. It must also include rules that govern all the usage of a language, which both can take considerable amount of time. The sentence to be generated must go through different rules that were set by the governing language to be able to complete a sentence. The method to generate artificial tweets proposed in this thesis reduces work for a data analyst by not constructing a context-free grammar to be able to produce a sentence.

A project for a twitter bot created by Tom Bowden to simulate tweets using markov chains can found from [42]. The method used in this project uses the words from the 200 latest tweets of a Twitter user and builds a markov chain based on these tweets. Markov chain is a stochastic model which shows a transition from one state to another state based only on the previous history by a probability. The method to generate artificial tweets proposed in this thesis is not limited to 200 latest tweets and does not create a markov chain for every Twitter user. The inherent nature of markov chain also does not allow the simulated tweet to contain words not used by the Twitter user that could possibly be used by the user since it can only create the markov-chain based on the words from the 200 latest tweets.

Another project to generate tweets is created by Max Woolf using a neural network can be found in [43]. This method trains a neural network with 7 layers using tweets from a Twitter user and context labels. The input to the neural network is only up to 40 characters. The quality of the generated tweets depends on the size of the dataset and to obtain the best generated tweets, it must be edited for the best results. Training a neural network can also take up a lot of time depending of the size of the dataset and the number of hidden layers. The method to generate artificial tweets proposed in this thesis does not require training a neural network with context labels. It is also not limited to 40 characters as input and it can generate tweets with words that the Twitter user may use but not used in the dataset.

3 Methodology

3.1. Analytic Modelling

Initially, real tweets were collected for the purpose of modelling the tweets. The Twitter API allows numerous information to be gathered when collecting the data from Twitter. However, not all the information is used in the experiments conducted in this thesis. The data or tweets to be fed into the simulator is formatted as shown in *Table 1*.

Table 1. Example format of the data gathered and to be used in simulator

FolloweeName	TweetID	TweetDate	UserID	FollowerName	TweetText
UserA	722196833803268620	2016-03-22 04:38:51	123456	UserB	RT @UserA: Do not stop retweeting this https://t.co/VnU1e3

The data from Twitter are formatted in the order of followee name, the ID of the tweet, the date and time the tweet was posted, the user ID of the tweet, the username of the user that posted the tweet, and the text contained in the tweet. The followee name is required in order to know what group or network the user or follower is from. This will assist in verifying the experimental results later to determine if the user is in the correct group. The tweet ID is required to uniquely identify each tweet. The tweet date and time is required for the purpose of knowing when the tweet was posted if the time the tweet was posted is relevant. The user ID is required to identify the user of the tweet since the name of the user is not unique. The name of the user or follower name is needed to determine who posted the tweet. Finally, the most important feature of the data is the text of the tweet. All the experiments in this thesis revolve around the text from the tweet and it must be present. The dataset collected for analysis in the next section is called dataset RU14k and it is detailed in section 4.2.

3.2. Fitting Distribution for tf-idf of Tweets

A distribution must be determined for the method to generate tweets using the *tf-idf* vectors of a user. The *tf-idf* vectors consist of *tf-idf* scores for all the unique terms in the given dataset so that each user in a dataset will have the same number of unique terms in their vectors. Different distributions were tested to fit onto the *tf-idf* vector based on the tweets of a Twitter user using the proprietary software Expertfit [44]. The distributions provided and tested in Expertfit were: Exponential, Gamma, Inverse Gaussian, Lognormal, Random walk, Uniform, and Weibull.

Weibull was chosen as the distribution because it was the top distribution based on the proprietary heuristics of Expertfit. The proprietary nature of the heuristics meant that the test statistics used are unclear since the help files of the software are unsupported due to the old age of the software.

After fitting the different distributions, the Anderson-Darling test was conducted. The Anderson-Darling test statistic is the weighted average of the squared differences between two distributions where the weights are largest for the right and left tails [19]. Many of the Twitter users from the RU14k with their *tf-idf* vectors were rejected by the Anderson-Darling test and the average error of the Weibull model mean relative to the sample mean was 53.25%. The Weibull distribution was kept as the model distribution even though the statistic tests from Expertfit did not agree with it. The reason for using the probability distribution is because for the purpose of simulation, it is better to use a distribution to model the nature of randomness of generated data instead of using an empirical distribution [19]. An empirical distribution cannot account for a population of data since it is only based on the sample of data provided. A distribution model can try to best represent a population rather than just a sample of data, but it cannot accurately represent the population [19]. The method to generate artificial tweets tries to generate words that a Twitter user may use related to the group the user is a part of, but the user has not used yet. With the use of a distribution model, this can help simulate the use of a word not used yet.

For the data collected in this part, a summary of statistics is shown in *Table 2*. The mean, median, and coefficient of variation (COV) are based on the tf-idf values from the words in the dataset RU14k for each user.

Table 2. Statistics of the Users used in Dataset RU14k

User	Mean	Median	COV	Model	Scale	Shape
A	2.79E-04	1.00E-08	26.14217	Weibull	2.29E-08	0.29175
B	5.79E-04	1.00E-08	12.58569	Weibull	7.05E-07	0.17642
C	0.00185	1.00E-08	3.82368	Weibull	1.39E-05	0.16094
D	9.78E-04	1.00E-08	7.4032	Weibull	1.60E-06	0.16436
E	0.0014	1.00E-08	5.12728	Weibull	2.52E-06	0.16219
F	5.74E-04	1.00E-08	12.69121	Weibull	7.49E-07	0.17597
G	6.38E-04	1.00E-08	11.41075	Weibull	6.81E-07	0.1737
H	0.0012	1.00E-08	6.00429	Weibull	8.09E-07	0.16578
I	4.67E-04	1.00E-08	15.62678	Weibull	2.55E-07	0.18591
J	0.00185	1.00E-08	3.81244	Weibull	1.74E-06	0.15679
K	0.00143	1.00E-08	5.02496	Weibull	1.78E-06	0.16055

In the following figures with the cumulative distribution function (CDF), the visual comparisons of the Weibull distribution with the empirical distribution are shown for 4 of the users in the dataset RU14k.

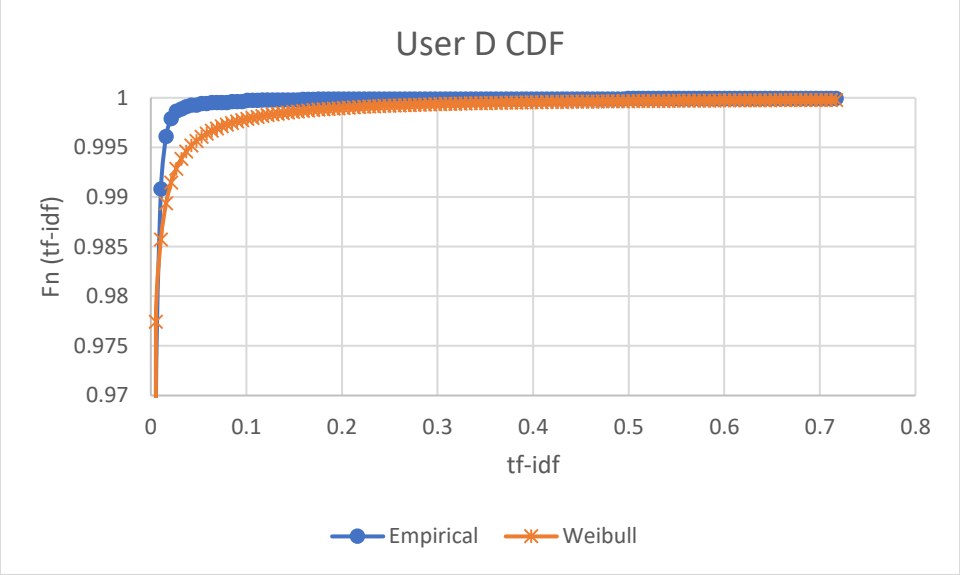

Fig. 1 The CDF for User D comparing Weibull and Gamma to the empirical distribution

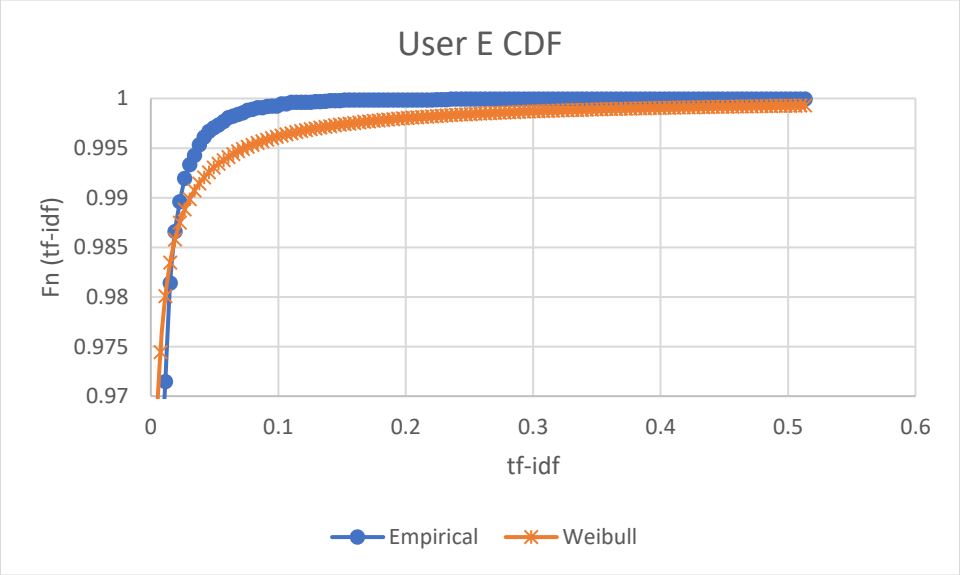

Fig. 2 The CDF for User E comparing Weibull and Gamma to the empirical distribution

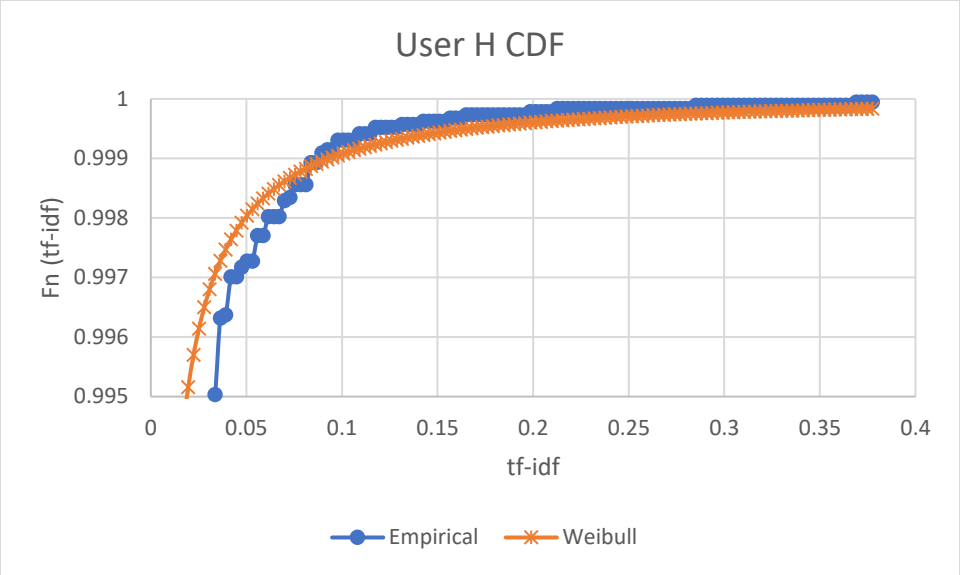

Fig. 3 The CDF for User H comparing Weibull and Gamma to the empirical distribution

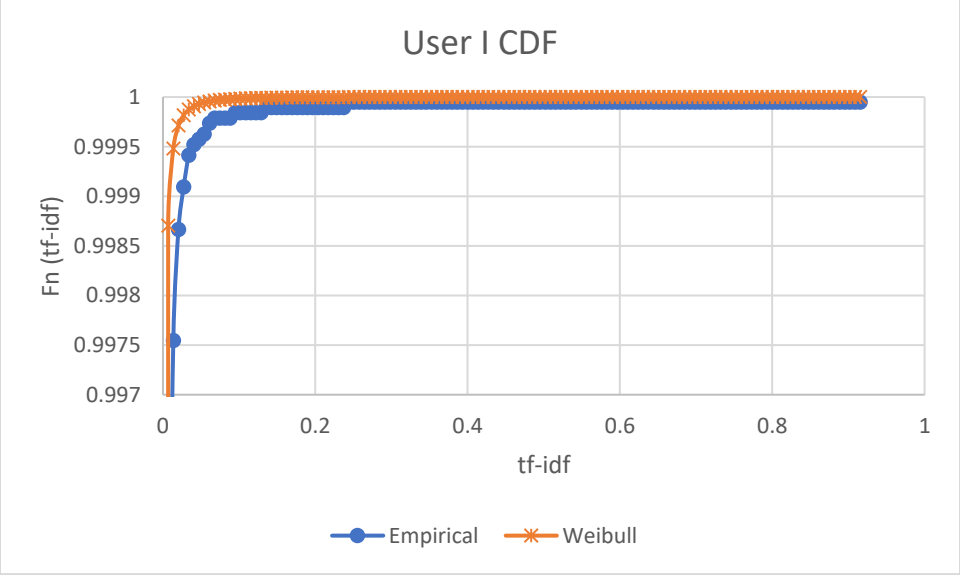

Fig. 4 The CDF for User I comparing Weibull and Gamma to the empirical distribution

It can be observed that there are a lot of zeroes present in the *tf-idf* data. The number of zeroes for the *tf-idf* of a Twitter user on average for the dataset RU14k is 84.86%. This means that on average 84.86% of the *tf-idf* of the words used by a user is 0. There are two reasons for the results of these zeroes. The first case is when the tf of a word not used by a user is 0, which results

in the *tf-idf* for the word to be 0 since the vector consists of unique terms from the whole dataset, which includes terms from other users. The second case is when there is a common word being used within a group. If there is a common word that is used amongst all the users in a group, the *idf* would result in 0 and provide *tf-idf* of 0 for the common word based on the definition of *tf-idf*.

It is based on the finding of this experiment that the Weibull distribution will be used in the generation method for tweets in the later section.

3.3. Algorithms Used in Preprocessing Data

3.3.1. Text Processing

The data collected must be processed to be usable in the experiments. Not all the information in the original text of the tweet is required to conduct the experiments. Algorithm 1 is the algorithm used for the text processing of the collected tweets. N is the number of tweets since every tweet goes through the algorithm, W is the set of terms in an unprocessed tweet and S is the set of stop words. The terms remaining after text processing is T.

Algorithm 1

Input: N tweets with unprocessed W terms
Output: Processed text of the N tweets with T terms

1: For each tweet:

2: Remove the term “photo:” or “photoset:” from the tweet text.

3: Remove the terms (RT @) from the tweet text.

4: Remove punctuations from the tweet text.

5: Remove URL links from the tweet text.

6: Remove special characters (non-alphabetical characters) from the tweet text.

7: Remove stop-words in S from the tweet text.

8: Change the tweet text to lower-case.

9: Accept the processed text if it has 3 or more terms

The text containing photos are removed from the tweets since this experiment does not consider image data. Retweets are not considered in these experiments as the experiment considers the individual tweets from a follower. The bag-of-words model is used so the grammar of the sentences are not considered in the experiment. The information from a URL link is usually in the source that the link directs to and thus it is not used. Only the alphabetical characters are kept so the symbol for hashtags (#) is removed but the word or phrase with no spaces associated with the hashtag is kept. Stop words are removed and then all the text is converted to lower-case. Finally, the processed text is accepted if it contains 3 or more terms since there should be enough information left over after processing to be used. The number of 3 or more terms was chosen because less than 3 terms may not have enough information to describe the context of the tweet. After text processing, T may have less than or equal to the number of terms in W. The text processing time complexity would be O(N * |W| * |S|).

3.3.2. Converting to tf-idf

Each tweet from a user is an individual document. To represent a user as a single document, all the processed tweets, which were individual documents belonging to a user, are compiled into a single document defined as an aggregated document [39]. U is the set of users since we aggregated all the user's tweets into a single document.

Algorithm 2

Input: Aggregated documents of terms in T for each user in U

Output: Aggregated document vectors of *tf-idf* for terms in T for each user in U

1: Initialize all document frequency of every unique term in T to 0.
2: For each unique term in T
3: Calculate the *df*
4: For each aggregated document of a user in U
5: For each unique term in T for an aggregated document
6: Calculate the *tf-idf*
7: Calculate the sum of the square of the vector components
8: Square root the sum of the square of vector components to get the vector magnitude

9: Normalize the aggregated document vectors of *tf-idf* to unit vectors by element-wise division by the vector magnitude

Once the tweets of a user are aggregated to a single document, Algorithm 2 can be applied to get the document vectors containing *tf-idf* values. The sum of the square of the vector components are also calculated for each document at the same time to save time and to be used in the next step of the algorithm. After going through all the documents, the vector magnitude is obtained by the square root of the sum of the vector components. The vector magnitude is used to normalize the document vectors of *tf-idf* to unit vectors to account for documents of different lengths [45]. This will also simplify the calculations for the next algorithm. The time complexity of Algorithm 2 is O(|U| * |T|) where T is the set of unique terms. If we assume a user's aggregated document to just be a tweet then O(N * |T|), where N is the number of tweets and T is the set of unique terms.

3.4. Tweet Generation

After the tweets have been processed and aggregated for each user, the data will be separated based on the followees and its followers. Each followee and its followers will form a group and they will have their own corpus containing the processed tweets. For each group, the following steps are repeated: After compiling a corpus for each group, the *tf-idf* vectors of the aggregated document for each user in the group is calculated using Algorithm 2. Next, the shape and scale parameters of a Weibull distribution for each user's vector is found using MLE. The average of the shape and scale parameters for the group is used to represent the shape and scale parameters for the group. The averaged shape and scale parameter will be used to generate artificial tweets for this group later. Before generating the tweets, the number of words to generate to represent a tweet must be known and word bins must be setup for each user. The number of words to generate to represent a tweet will have a minimum of 3 words and a maximum number of words based on the median number of words in a tweet after text processing for a group. The word bins are setup based on the unique *tf-idf* values for each user. A bin for each unique *tf-idf* value will contain all the words that the user has used that equals the *tf-idf* value in the *tf-idf* vector representing the user. After all the bins are setup for each unique *tf-idf* value of each user in the group, it is ready to generate artificial tweets.

To generate an artificial tweet, assumed to be after text processing, the number of words in a tweet will be randomly generated using a uniform distribution based on the minimum to the maximum number of words obtained from the previous step. Next, to generate a word for a tweet, the averaged shape and scale parameters of a group are applied to a Weibull distribution to get a value X that represents a *tf-idf* value. The bin assigned with the *tf-idf* value closest to X is selected. The closest bin is determined by the absolute difference between X and the *tf-idf* value of a bin. If there is a tie for the closest bin, the first bin chosen for the minimum distance is selected as the closest bin. From the bin, a word is randomly selected based on a uniform distribution and it will be selected to be a word in the artificial tweet. The choice of uniform distribution is arbitrary, but it is a simple distribution that works to choose a random number of words for a tweet that makes computation simpler. This process is repeated until the artificial tweet has reached the initially determined required amount of words.

The desired number of artificial tweets is determined by the creator of the generated dataset. By default, the generation method will create the number of artificial tweets for the user in the generated dataset to be the same as the real dataset. To create a larger dataset, it can scale up a user's number of tweets by a factor to reach a desired number of artificial tweets. After generating the desired number of artificial tweets, the artificial tweets are compiled together to represent the corpus of a simulated group. Then the corpus of these simulated groups with artificial tweets are compiled together as well to form a single corpus. This single corpus will be considered a generated dataset with artificial tweets to simulate the original dataset.

3.5. Recommendation Algorithm

The last algorithm of the recommender system in this thesis is the recommendation algorithm. Algorithm 3 consists of k-means clustering of documents [25], [46]–[49], followed by computing a list of scores for top-R recommendations. R is a number from 1 to the number of other users in the dataset but 10, 15, 20, and 25 were used in the experiments.

Algorithm 3

Input: k number of clusters, userID of user to get recommendation, number of top-R

Output: k clusters of aggregated documents, top-R similarity score list for userID

1: Select k random aggregated documents as centroids for k number of clusters
2: Until convergence:
3: For each aggregated document:
4: For each cluster:
5: Calculate the cosine similarity of the aggregated document to centroid of cluster
6: Assign aggregated document to cluster with the greatest cosine similarity to centroid of cluster
7: Calculate new aggregated document centroid of cluster
8: For each k cluster
9: For each aggregated document with p = 0 until p < U-1 in the cluster
10: For each aggregated document with q = p+1 until q < U in the cluster
11: Calculate the cosine similarity between aggregated documents in cluster k
12: Use the cosine similarity as scores between aggregated documents in cluster k
13: Create a list of scores for userID based on the cosine similarity to other users
14: Sort the similarity score list by descending order
15: Present the top-R similarity score list to userID

After Algorithm 1 and Algorithm 2 are applied, Algorithm 3 is applied on the processed aggregated documents. The algorithm begins by choosing random k aggregated documents to be represented as the centroid aggregated document of each k clusters, the userID of the user that is getting a recommendation, and R for the number of top-R recommendations. The centroid in this sense is an aggregated document vector to represent the cluster, which represents a group of users with similar interests i.e. a followee of a group like RyersonU, TheCatTweeting, theScore, TorontoStar, and weathernetwork. Each aggregated document is assigned to a cluster based on the cosine similarity as the distance measure used in the k-means clustering algorithm based on lines 1-7 of Algorithm 3. The cosine similarity is used because of the sparsity of the document vectors. When all aggregated documents have been assigned to a cluster for an iteration, a new aggregated document centroid is calculated. The aggregated document centroid is calculated by taking the

mean of each term of all the aggregated document vectors in the cluster. However, the mean aggregated document vector may not be normalized after calculating the mean and thus the newly calculated mean aggregated document vector must be normalized by dividing by the magnitude of the mean aggregated document vector. The normalized mean aggregated document vector becomes the new centroid of the cluster of aggregated documents. Convergence is met when there are no changes to the members in the clusters and the clusters remain the same, which means the objective function is met.

After the k-means clustering has been completed, the score between users can be calculated based on lines 8-12 in Algorithm 3. A user is represented as an aggregated document and thus the cosine similarity between aggregated documents can be used as a score [17]. The higher the cosine similarity, the higher the score and the two aggregated documents or users would be considered more similar. The cosine similarity is calculated between every single pair of aggregated documents in every k clusters. A list of cosine similarity score that is sorted in descending order can be found for the user with userID. With the list of cosine similarity scores for the user, the top-R users in the list will be provided as a recommendation. The top-R users will be determined from the other users present in the cluster that the user with userID is found in because the user with userID can only be in 1 cluster at a time.

The k-means clustering is $O(|U| * k * |T| * i)$ where U is the set of users since we aggregated the user's tweets into a single document, k is the number of clusters, T is the set of unique terms, and i is the number of iterations for k-means. The time complexity is governed where each iteration each user must check each cluster to find the most similar cluster that is computed on each unique term based on cosine similarity. Similarly, if we assume a user's aggregated document is a tweet then it is $O(N * k * |T| * i)$ where N is the number of tweets.

The scoring of the users is $O(|U|^2 * k * |T|)$ where U is the set of aggregated documents in a cluster, k is the number of clusters, and T is the set of unique terms in a document. Each cluster has each user compute each unique term with another user for this time complexity. Overall for the recommendation algorithm, it is $O(|U|^2 * k * |T|)$. If we assume a user's aggregated document to just be a tweet then the overall time complexity for the recommendation algorithm is $O(N^2 * k * |T|)$, where N is the number of tweets.

3.6. Multi-Agent Model

To be able to simulate a distributed environment and Twitter users, multi-agent modelling is used. Multi-agent models are effective tools for simulation of complex environments such as distributed computing, recommender systems, and user communications used in this work. The multi-agent model for the conducted experiments is composed of the controller agent, organizing agent, user agent(s), recommender agent(s), starter agent, data crawler agent, and user generation simulation agent. Each agent plays a part in the whole model and each agent will be used in the experiments. The agents work together to achieve the objective of providing a recommendation to a user who to follow given a dataset. The controller agent is an interface for a user using a simulated recommender system. It creates the other agents, with each agent containing their own tasks, listed previously to run a simulation. The nodes mentioned in the experiments are the recommender agents which represent a processing component in a distributed system. When there are more than 1 node for the experiment, there are multiple recommender agents with each representing a node. Each node will run through the algorithms separately i.e. each node will run through Algorithms 1-3. The whole distributed system is simulated based on these different agents representing a different part of the system.

The controller agent oversees everything from creating all the different agents to controlling the flow of the simulation by directing when to collect data, start a user generation simulation, or starting a simulation to measure scalability performance. The controller agent reads in different parameters given by a human user and then executes the task required. The parameters of the followee name(s) and the number of followers are provided by the controller agent to the data crawler agent. The parameters of the number of the artificial tweets to generate and where the corpus is located is provided by the controller agent to the user generation simulation agent. The parameters of the number of nodes, number of top recommendations, text processing parameters (removing hashtag symbols, retweets, top words), the location of the corpus or dataset in the system, and the recommendation algorithm, like Algorithm 3, to use in the simulation are provided to the controller agent to create a simulation for performance measurement. The number of nodes determines the number of recommender agents to be created by the controller agent. The controller agent also determines how the tweets are distributed between the recommender agents. The users' tweets are distributed as evenly as possible to the number of recommender agents by sending a

user's tweets to the recommender agent with the lowest total number of tweets until every user's tweets have been distributed. The tweets of the user to receive a recommendation is copied to every recommender agent when there are 2 or more nodes so that the user receiving a recommendation can be clustered with the other users in the other recommender agents.

The data crawler agent collects tweets from Twitter using the Twitter API based on the parameters given by the controller agent. The data from Twitter is collected in the format explained in section 3.1. The user generation simulation agent creates a simulation where a generated number of artificial tweets is created and a k-means clustering of the users' tweets is provided in the end to check the clusters for verification.

The organizing agent is responsible for ensuring that the recommender agent has the correct number of user agents and the recommendation list that is given to the user agent that requests a recommendation. When each user agent is created, the user agent will communicate to the organizing agent that it is ready. The organizing agent communicates to the recommender agent(s) when all the user agents are ready. This ensures that the correct number of user agents are communicating to the recommender agents. If there are more than 1 recommender agent, it is the responsibility of the organizing agent to merge the recommendation lists from each recommender agent into a single list. The recommendation list is provided by the organizing agent when it is available to be queried by a user agent.

The user agent is used to represent a user of Twitter. This agent holds the tweets of the user whether it is the artificial tweets, or the real tweets collected from Twitter. Upon creation and initialization of the user agent, the user agent will notify the organizing agent that it is connected to a recommender agent. The tweets of the user agent will send messages containing the tweet text to one or more recommender agents depending on the configuration of the simulation. Once it has finished sending all its tweets, it will notify the starter agent of the tweeting completion.

The recommender agent is responsible for running the algorithms explained in section 3.3 and 3.5. Tweets sent by the user agents are received by the recommender agent as well. The measurement of communication cost that comes from message passing is also measured by this agent. The recommender agent handles the timing of the algorithms as it runs through them. If a user agent does not have any substantial text in the tweets after text processing from Algorithm 1, the recommender agent(s) will not allow the user agent to query for a recommendation and remove

the user agent from the list of agents that are supposed to communicate with the recommender agent(s). Each recommender agent is responsible for the recommendation list that a user agent requests and if there are multiple recommender agents, each recommender agent will send their recommendation list to the organizing agent to merge into a single list.

The starter agent is responsible for each simulation iterations. It will start a simulation once all the other agents are ready. The recommendation algorithm is also started by the starter agent when the recommender agent(s) have completed Algorithm 1 and Algorithm 2. The user agent(s) are notified that querying for a recommendation is allowed by the starter agent once the recommendation algorithm is complete.

A summary of the messages passed between the agents in the multi-agent system model can be found in *Table 3*.

Table 3. A summary of the messages by the agent sender and the agent receiver of a message.

Sender	**Message**	**Receiver**
Recommender	Received Last Tweet	User
	Message Passing Cost	Starter
	Denied Querying	User
	Remove Users From Total	Starter
	Text Processing Complete	Starter
	Tweeting TFIDF Algorithm Calculation Completed	Starter
	Merge Score Lists	Organizing
Organizing	Update user list	Recommender
	Query Available	Starter
User	Connected to Rec Agent	Organizing
	Tweeting Completed	Starter
	Tweet From User Agent	Recommender Agent
Starter	Start Simulation	User
	Start Recommend Algorithms	Recommender
	Start Querying	User

3.7. Flow of Simulation

The experiments conducted in this thesis use the multi-agent model described in section 3.6. The timings of the experiments are calculated from the start of the first line of code for each process to the last line of code. Depending on the amount of data processed, each process can have

different timings. The performance measurement experiments for the recommendation algorithm involving k-means clustering are timed based on the algorithm from section 3.5. They are timed based on when the recommendation algorithm starts to when the recommendation algorithm ends. The performance measurement timing for total recommendation time is from when the starter agent starts the simulation to the end of the simulation to obtain the total simulation time. The total recommendation time is composed of the timing for data allocation, the timing for text processing (Algorithm 1), the timing for *tf-idf* transformation (Algorithm 2), the recommendation algorithm (Algorithm 3), and the merge time for the results of recommendation algorithm. Each recommender agent runs through the 3 algorithms then merges the recommendation lists at the end if there are multiple recommender agents otherwise it is a single recommendation list with no merging required. A summary of the simulator running through the experiments can be found in *Fig. 5* and *Fig. 6*.

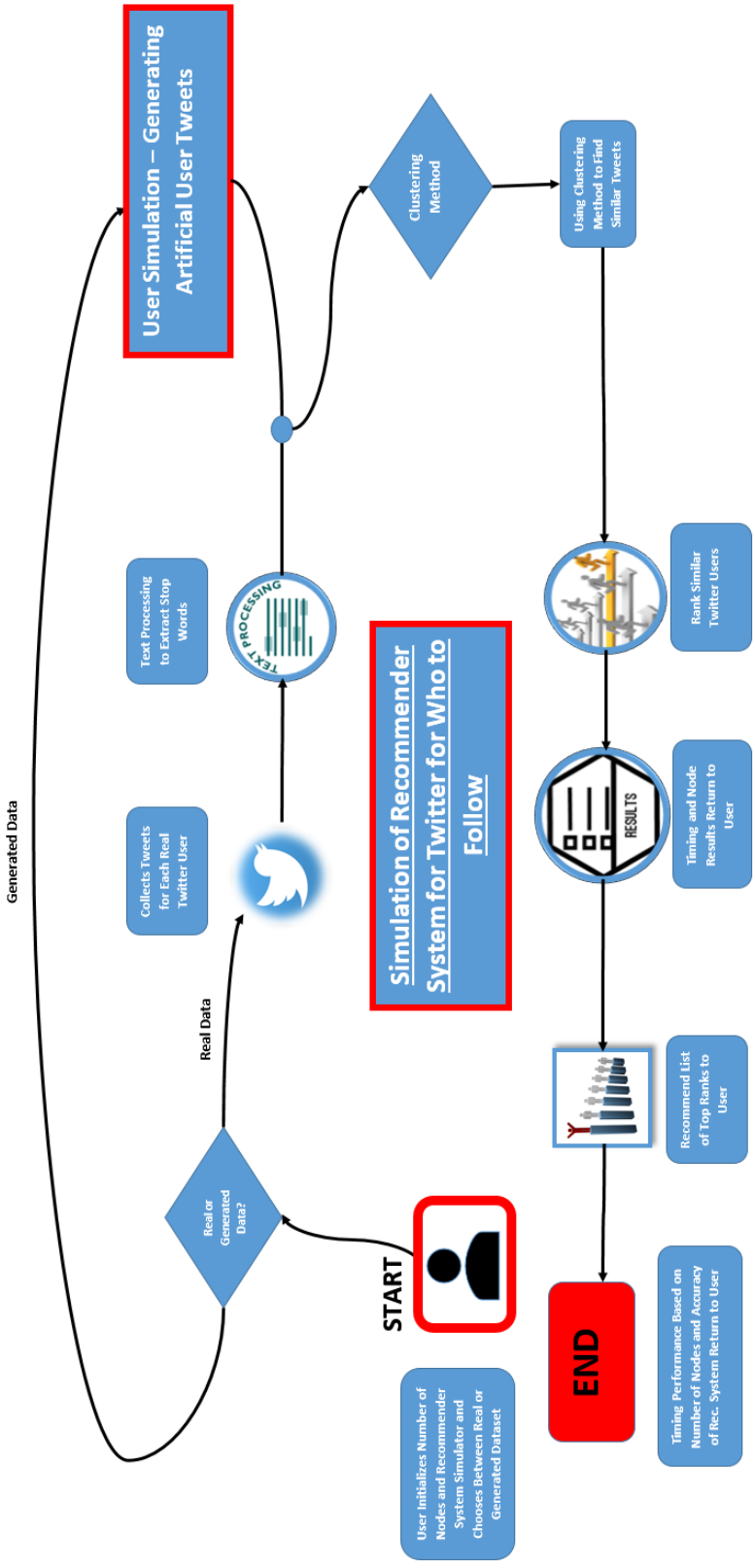

Fig. 5 Flowchart of the simulator of a recommender system for Twitter

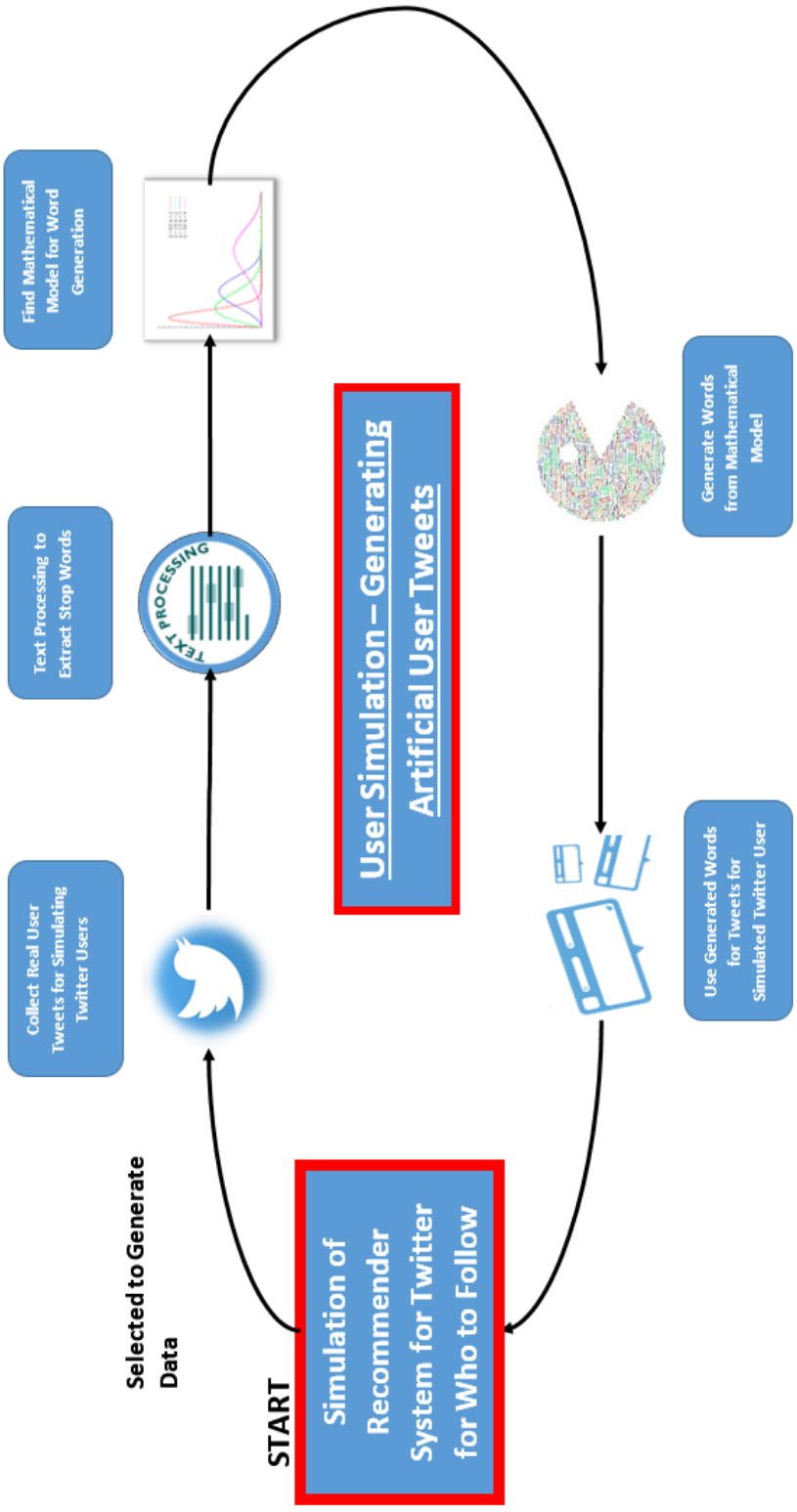

Fig. 6 Flowchart for the generation of artificial tweets

4 Implementation and Results

4.1. Introduction

In this chapter, the datasets and the experiments conducted will be covered. The datasets compiled for the experiments will be explained. The implementation for the experiments will use the methodology covered in Chapter 3. Finally, the results of the experiments will be analyzed.

4.2. The Datasets

In the experiments, there will be eight different datasets used. For the datasets, 5 followees (RyersonU, TheCatTweeting, theScore, TorontoStar, weathernetwork) were used to collect the tweets of its followers and the followee itself. The seed users for the datasets are followees mentioned previously and they were randomly selected based on publicly accessible data (regarding their followers and tweets). The seeds are selected to capture diversity in the users' interests of followers. For example, RyersonU is selected as the followee for those interested in RyersonU, TheCatTweeting is a followee who is followed by the people interested in cats, weathernetwork is followed by the people interested in the weather, TorontoStar for those interested in the news, and theScore for people interested in sports. It is entirely possible there are overlaps in interests between followers of these 5 selected seed users, but during the data collection, the followees chosen did not have the followers overlapping. Each follower is assumed to be part of the group of one followee since they were found from the followee's follower list when collected. All the tweets were gathered over a 5-month period at midnight for maintaining a consistent time of when to collect the tweets. The tweets of the latest 20 followers for each day, provided by the Twitter API, of each respective followee were collected only if the following criteria were true: the minimum number of tweets is 1, the follower must have a public account, and the follower must not have been collected already from a previous day. The data that did not meet the previous criteria were not kept for the dataset. The assumption is that if the follower was not present in the previous midnight check, it is a new follower and only the tweets from the day before joining are kept. The tweets before following the respective followees are used for analysis. The collection of tweets was implemented by using a Java library, Twitter4J [50]. In the first dataset RU14k, it is compiled from tweets taken from 13 random users that follow RyersonU and the tweets from RyersonU itself. In the second dataset GEN_RU14k, it is a generated dataset based on dataset RU14k using the methodology discussed in Chapter 3. The users in the generated dataset

is kept the same as the users in the real dataset. Thus, the followee and the number of followers in the generated dataset is the same as its counterpart. The number of generated tweets for each generated user is kept consistent to the number of tweets the user had in the dataset's counterpart. The third dataset ALL424k contains the 5 followees mentioned previously and 52 followers of each followee. The followers were chosen in a way so that we can keep the data balanced. This means that the total number of tweets from each followee's group has approximately the same total number of tweets. Based on this approximate total, it was found that 52 followers from each group would reach this approximate total when the number of tweets of each user were sorted in descending order. Similar to GEN_RU14k, GEN_ALL424k is a generated dataset of ALL424k. GEN_ALL424k_TO_1200k is a generated dataset of ALL424k that is scaled up to around 1.27 million tweets by increasing the number of tweets of each user in ALL424k by a factor of 3. The summary of the datasets is in *Table 4*.

Table 4. Datasets used for the simulations

Dataset	Tweets	Unique Words	Average Number of Tweets
RU14k	14,265	18732	1023
GEN_RU14k	14,312	18290	1023
ALL424k	424,431	264501	1652
GEN_ALL424k	424,459	248144	1652
GEN_ALL424k_TO_1200k	1,273,282	264465	4955

4.3. User Generation

The method explained in Chapter 3 were implemented in the simulator. Datasets RU14k, and ALL424k were used for the generation of tweets. The simulator is given the number of followees, followers, and the number of tweets to be generated. Using the method from Chapter 3, a separate program written in Java was used to generate the *tf-idf* matrix of the words present in a dataset containing real tweets for each user. The *tf-idf* values for each user were fit with various distributions to find the best fit. The distribution with the best fit was chosen to be the distribution

used to generate the words in the experiment. The experiment generated the same number of tweets for the generated user as the real user for all the users in the dataset. The simulator runs k-means clustering, with k = 3 as the cut-off value that is commonly used, on the real users and the generated users separately.

4.4. The Testing Environment

The simulations were run on a 4.0 GHz i7-4790k with 4 cores desktop computer with 16 GB RAM. The OS the machine used was Windows 10. Java was installed on the machine to be able to run the simulator developed in Java. The performance, cost, and clustering results were obtained from running the simulations on this machine. The recommender system simulator was created in Java with the middleware JADE (Java Agent Development Framework) that uses the FIPA (Foundation for Intelligent Physical Agents) standards [51]. An agent created by JADE is a thread in Java. JADE also handles the communication of the agents.

4.5. Results

The results will show the performance improvement and communication cost of the recommender system simulator in terms of scalability by using multiple nodes. A recommender system has been simulated to be used for both the real and artificial data to test and validate how artificial data is similar to real data.

4.5.1. Simulation of Recommender System

The simulated recommender system identifies similar users based on the similarity between the aggregations of each users' tweets. Similarity between the aggregated tweets can be identified by the k-means clustering method followed by the list of similarity. According to Weng et al., there is similarity between the tweets of the users in each group when they form groups in social networks, it is called homophily [52].

The simulator consists of a controller agent, an organizing agent, a starter agent, user agents, and recommender agent(s). All the datasets were used for the experiments in this thesis. The simulator ran on different configurations. The configurations consisted of changing the number of nodes, which a recommender agent represents a node, to test the scalability and accuracy. Text processing of removing hash-tags symbols, stop-words, and retweets from tweets were consistently used for all nodes in the configuration.

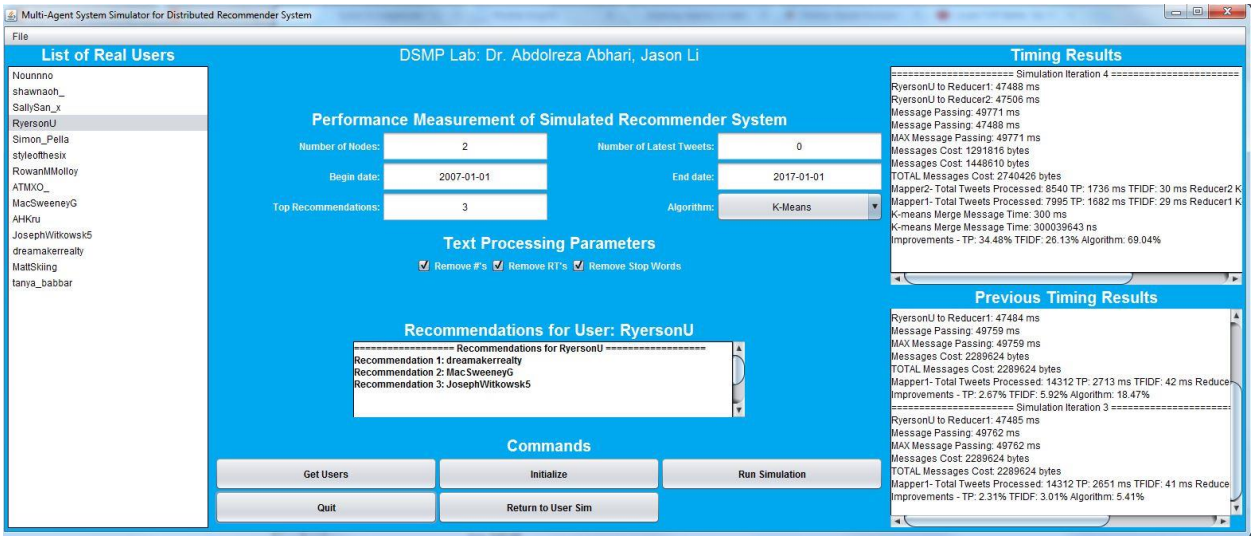

Fig. 7 Screenshot of the simulator used in the experiments

4.5.2. Validation and Verification of the Recommender System Simulator

The validation of the simulator was completed by using the validation technique of Degenerate tests [53], [54]. The input and internal parameters were changed, and the results were observed to ensure it was as close to the expected results of a real-life situation. The inputs given would be the datasets that were provided, increasing in size, and the internal parameters being tested on would be the number of nodes for the simulator. As we increase the number of nodes, the timings on the algorithms should decrease. In the following results, it is observed that there is an improvement in the performance as the number of nodes increase and thus the timings of the algorithms are decreasing. Therefore, the simulator was validated by the results provided by the simulations.

The verification of the simulator was completed by looking at the message passed by the agents [55]. In JADE, there is a Sniffer Agent that is able to read the messages sent and received by the agents in the multi-agent environment [56]. The Sniffer Agent was used to read the messages sent and received by the different agents in the system. A log was kept of the messages the Sniffer Agent observed and it was compared to the message logs that were manually logged. The Sniffer Agent verified the flow of the messages sent and received by the different agents created for the simulation.

4.5.2.1. Performance of Recommender System by Using Real and Artificial Data

The scalability test of a recommender system using a simulator was conducted in these experiments. The performance improvements in the following results are with respect to the running time of the recommendation algorithm used in the recommender system simulator. It is calculated by taking the ratio of the timing result as the number of nodes increase from 1 node. X in the formula can be 2, 4, or 8 nodes.

$$\text{Performance Improvement} = \frac{\text{Timing of 1 Node} - \text{Timing of X Node}}{\text{Timing of 1 Node}} * 100 \quad (13)$$

In the experiment for dataset RU14k and GEN_RU14k, the performance improvements from 1 node to 2 nodes, 1 node to 4 nodes, and 1 node to 8 nodes were compared. In the dataset RU14k a performance improvement of 56% from 1 to 2 nodes, 89% from 1 to 4 nodes, and 97% from 1 to 8 nodes. Similarly in the generated dataset GEN_RU14k, the observations were 47%, 86%, 99% for 1 to 2 nodes, 1 to 4 nodes, and 1 to 8 nodes respectively.

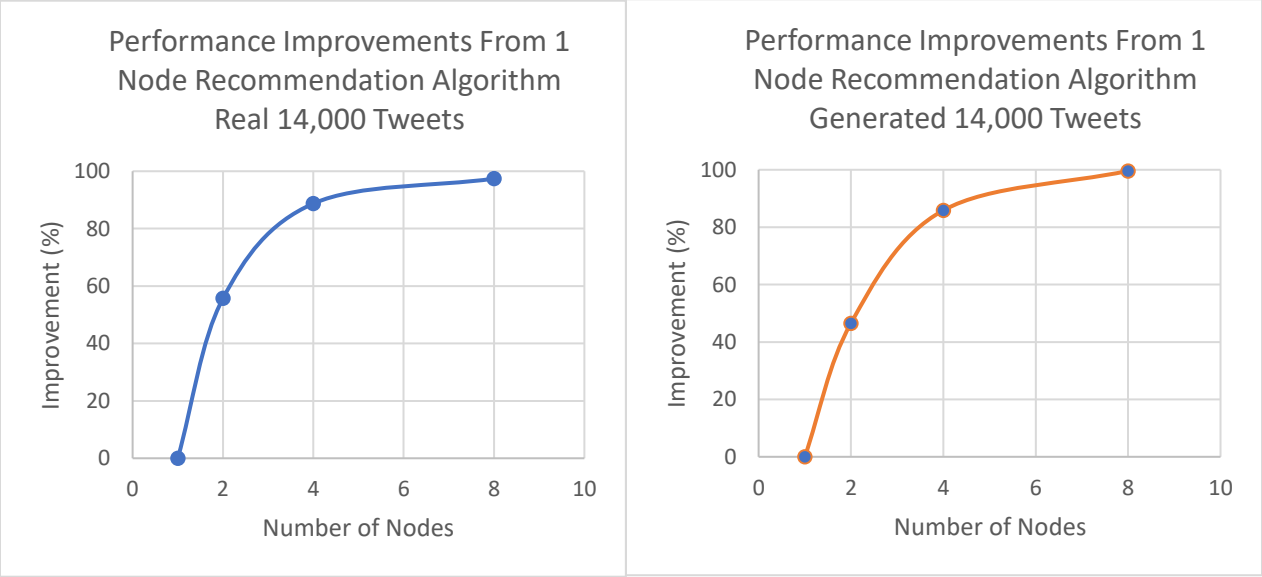

Fig. 8 Performance improvements of k-means clustering from 1 Node for RU14k and GEN_RU14k.

In the experiment for the performance improvements of a larger dataset ALL424k and the generated dataset GEN_ALL424k, the results of performance improvements from 1 to 2 nodes, 1

to 4 nodes, and 1 to 8 nodes can be seen. From the experiment on ALL424k, a performance improvement of 60%, 81% and 89% were observed. Similarly for the generated dataset GEN_ALL424k, a performance improvement of 51%, 79%, and 90% were observed.

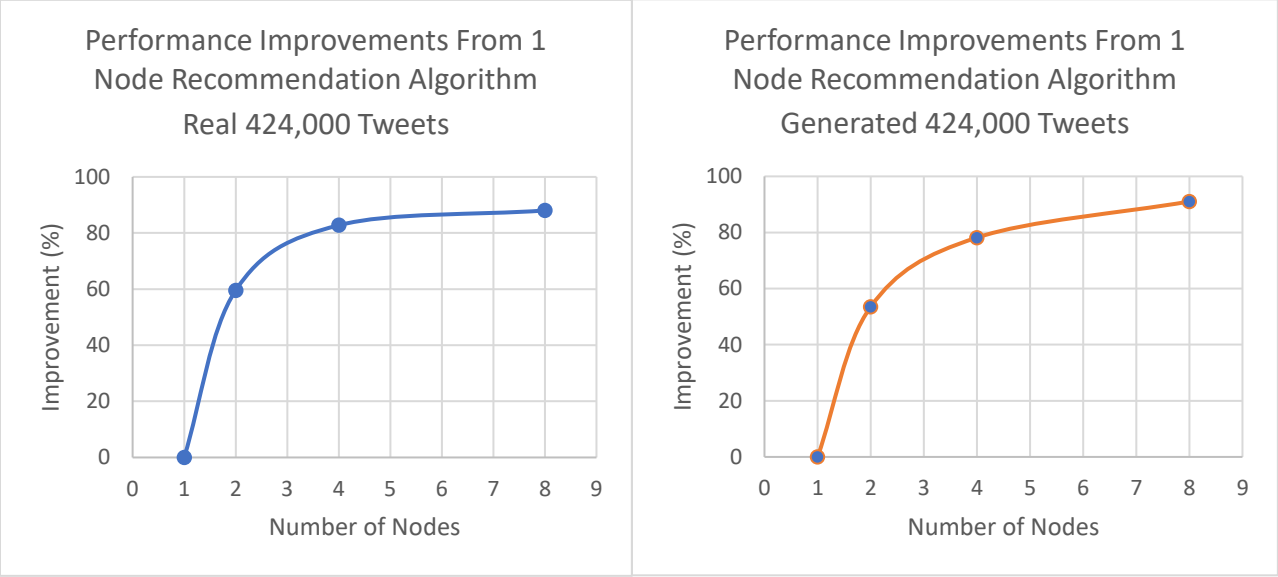

Fig. 9 Performance improvements of k-means clustering from 1 node for ALL424k and GEN_ALL424k

4.5.2.2. Cost of Recommender System Scalability Simulation

Communication cost is the overhead of using a distributed system. Even for a single node, there is some sort of communication cost associated with it. The communication cost is measured by the following equation:

$$\text{Communication Cost} = \sum_{i} \text{agent(i) messages to a node} \tag{14}$$

The results in *Fig. 10* and *Fig. 11* show the communication costs for RU14k and GEN_RU14k.

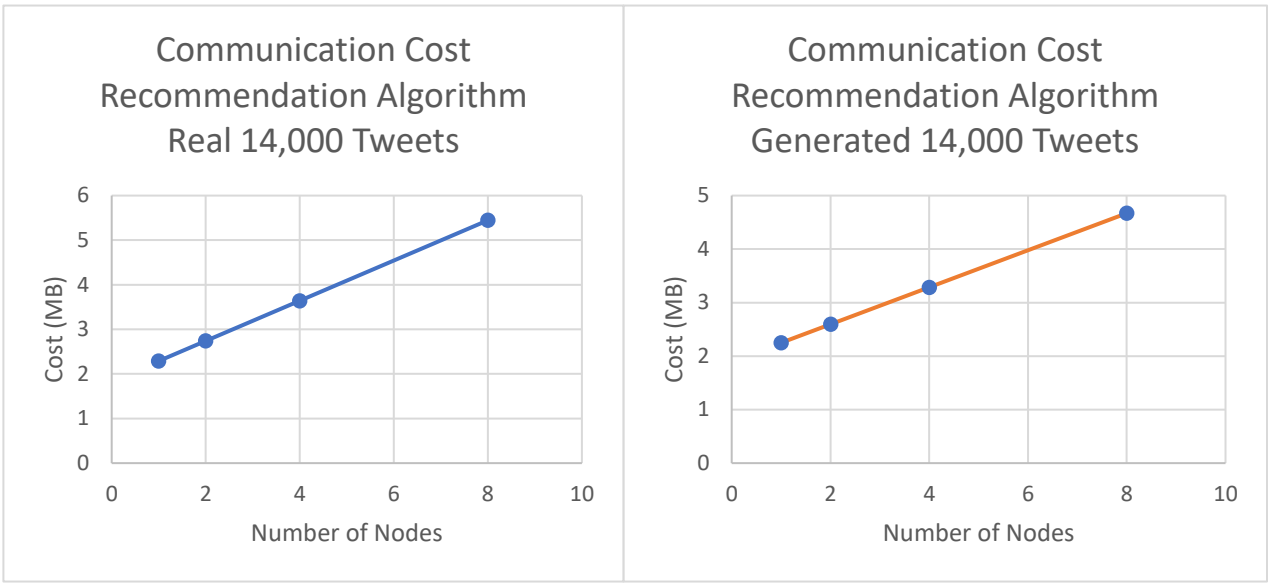

Fig. 10 Communication Costs for RU14k and GEN_RU14k.

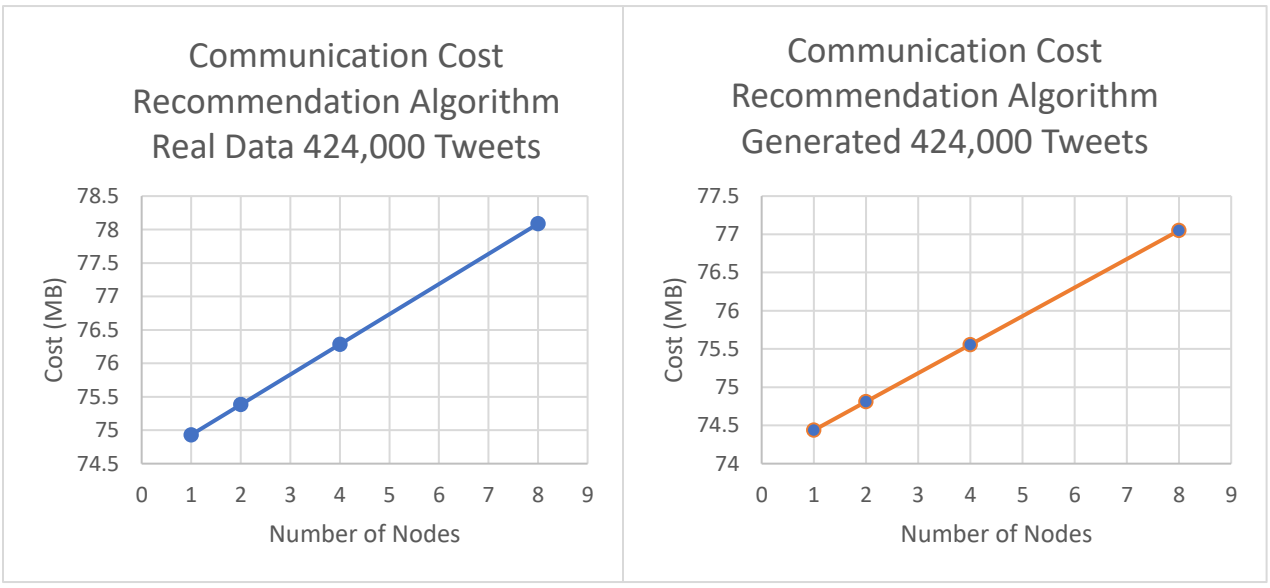

Fig. 11 Communication Costs for ALL424k and GEN_ALL424k.

The communication cost remains similar in the trend for the experiments of the real and generated datasets. The communication cost can be seen as linear with the cost increasing as the number of megabytes with the number of nodes increasing.

4.6. Scalability Test

In the experiment for dataset GEN_ALL424k_TO_1200k, there were no real data to be associated with it. However from the previous two experiments, we can confirm that the generated data is similar to real data. Therefore we can assume that the result of this data would be similar to real data. In this experiment we observed a performance improvement of 59%, 81% and 90% from 1 to 2 nodes, 2 to 4 nodes, and 4 to 8 nodes as shown in *Fig. 12*. The communication cost for GEN_ALL424k_TO_1200k was also observed to be linear similar to the communication cost of the smaller datasets. *Fig. 13* shows the communication cost for GEN_ALL424k_TO_1200k.

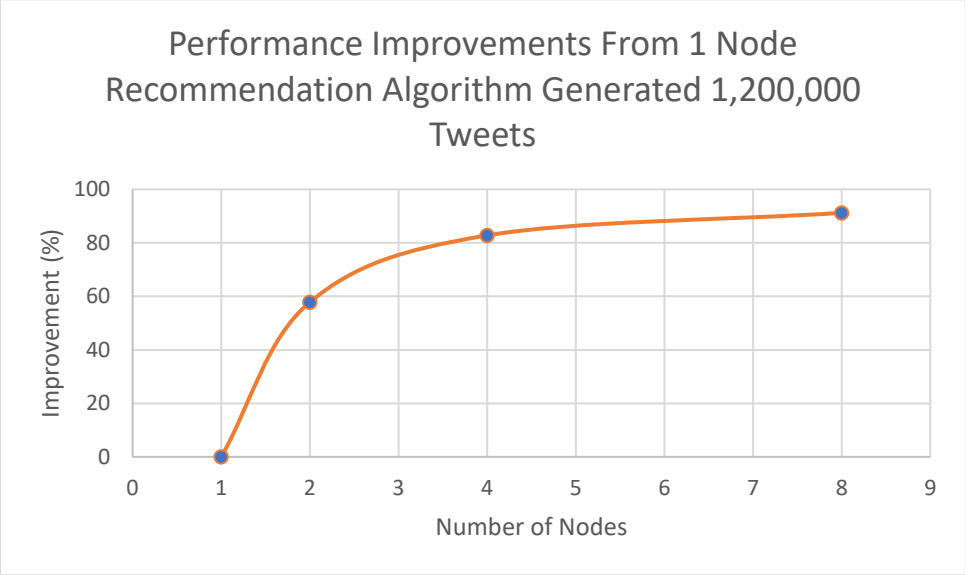

Fig. 12 Performance improvements of recommendation algorithm from 1 node for GEN_ALL424k_TO_1200k

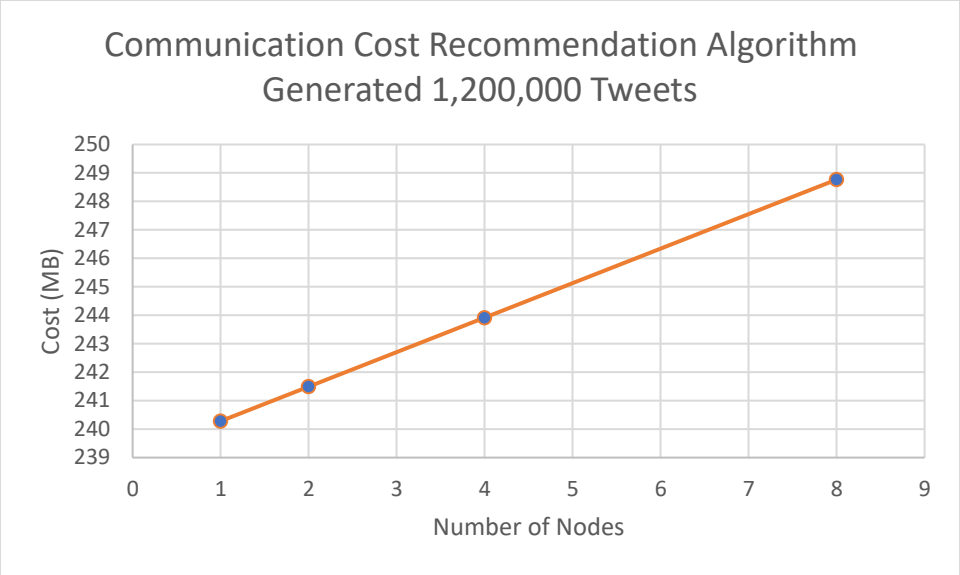

Fig. 13 Communication Costs for GEN_ALL424k_TO_1200k

4.6.1. Evaluation of Accuracy of Recommender System Simulator

The mean absolute error (MAE) was used to evaluate the accuracy of the recommender system simulator. MAE is defined in the following equation:

$$MAE = \frac{\sum_{i=1}^{n} |e_i|}{n} = \frac{\sum_{i=1}^{n} |r_i - x_i|}{n} \quad (15)$$

where r is the recommended value and x is the actual value. This measurement gives the recommendation accuracy where 0 is considered perfect with no errors and 1 is considered the worst with error in every single recommendation. In this experiment the values are evaluated based on if the user is a follower of RyersonU. This is a special case to test the accuracy since the user to get a recommendation is RyersonU. To be able to verify the accuracy of the results, RyersonU was chosen since the tweets of each follower of RyersonU was collected before following RyersonU so that it can confirm the recommendation will provide an actual follower of RyersonU. If the value of r is 1, it means that it is recommending this user to follow RyersonU while a value of 0 is recommending this user to follow a user that is not RyersonU. For x, a value of 1 means the user is actually a follower of RyersonU and a value of 0 means the user is not actually a follower

of RyersonU. In a list of top n, each user in the list is checked if the user is a follower of RyersonU. The following *Table 5* shows the MAE results as the number of nodes and top n increases for dataset GEN_ALL424k_TO_1200k.

Table 5 MAE of the Recommender System Simulator

Node(s)	top 10	top 15	top 20	top 25
1	0.666667	0.6	0.616667	0.626667
2	0.7	0.666667	0.7	0.68
4	0.666667	0.6	0.6	0.653333
8	0.6	0.633333	0.6	0.64

For 1 node, the range of MAE is 0.667 to 0.6 for top 10 to top 25. For 2 nodes, the range of MAE is 0.7 to 0.667 for top 10 to top 25. For 4 nodes, the range of MAE is 0.667 to 0.6 for top 10 to top 25. For 8 nodes, the range of MAE is 0.64 to 0.6 for top 10 to top 25. From the results, it can be seen the results are relatively consistent as the number of nodes and top n increases.

4.6.2. Total Recommendation Time

The total recommendation time consists of data allocation, Algorithms 1, 2, 3, and the merge time, if the configuration is more than 1 node, of the recommendation lists. In *Fig. 14*, the performance improvements of total recommendation time for RU14k and GEN_RU14k can be found. For RU14k, the performance improvements are 14%, 23%, and 24% for 1 to 2 nodes, 1 to 4 nodes, and 1 to 8 nodes respectively. While the performance improvements for GEN_RU14k are 22%, 42%, and 53%. The performance improvements of total recommendation time for ALL424k and GEN_ALL424k can be found in *Fig. 15*. The performance improvements are 53%, 74%, and 86% in ALL424k for 1 to 2 nodes, 1 to 4 nodes, and 1 to 8 nodes respectively. In GEN_ALL424k, the performance improvements are 33%, 72%, and 86%.

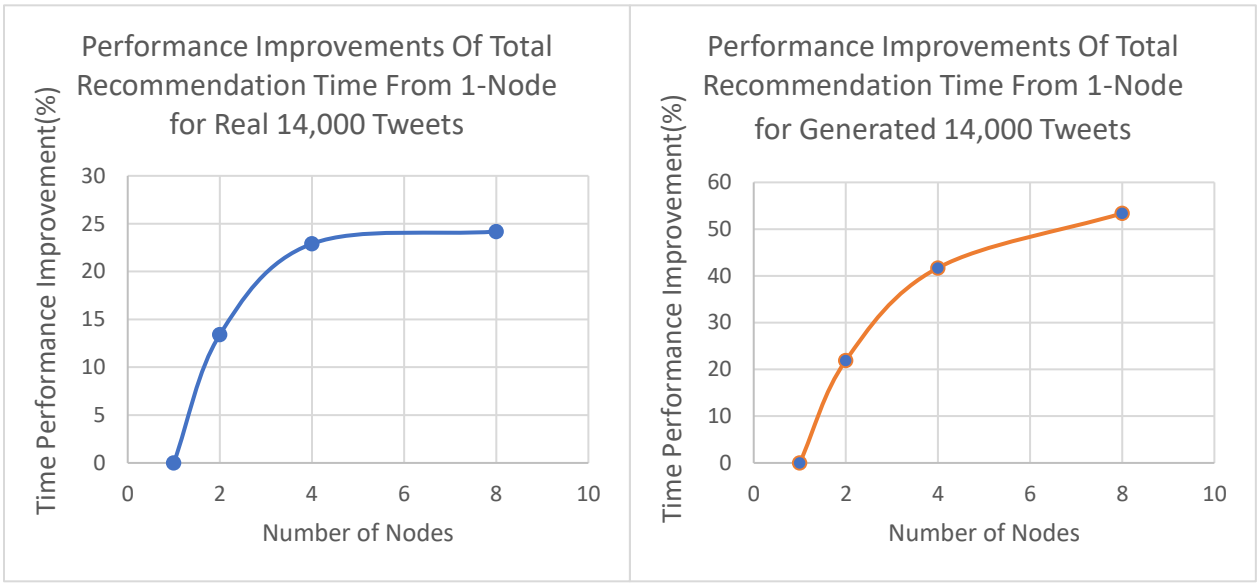

Fig. 14 Performance improvements of the total recommendation time from 1 node for RU14k and GEN_RU14k.

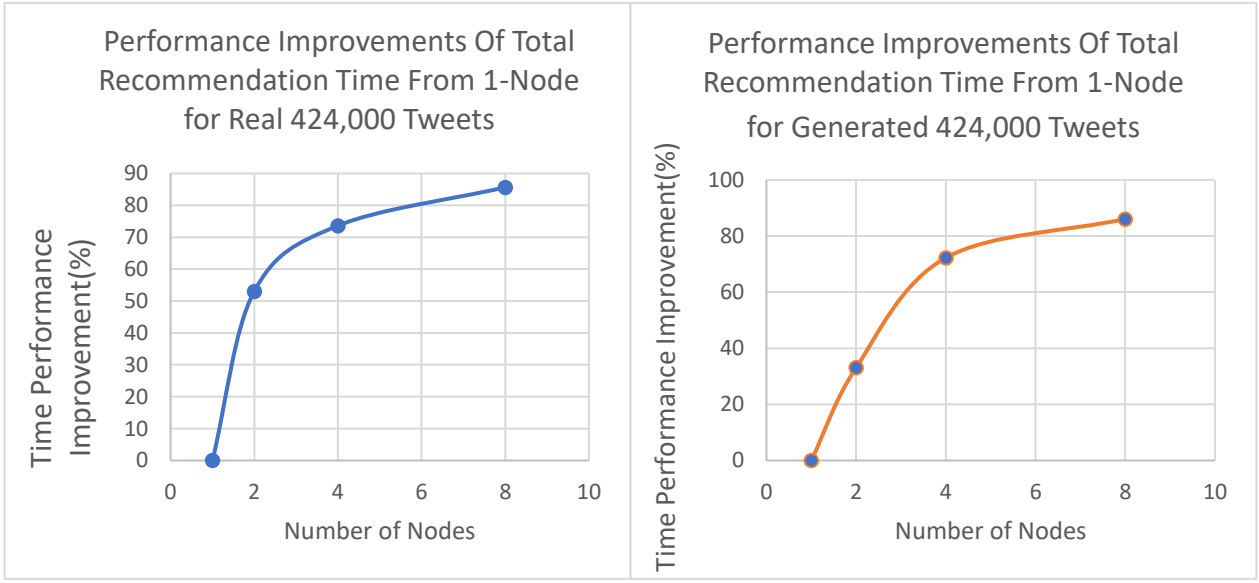

Fig. 15 Performance improvements of total recommendation time from 1 node for ALL424k and GEN_ALL424k.

Finally, for GEN_ALL424k_TO_1200k in *Fig. 16*, the performance improvements are 30%, 47%, and 55% for 1 to 2 nodes, 1 to 4 nodes, and 1 to 8 nodes respectively.

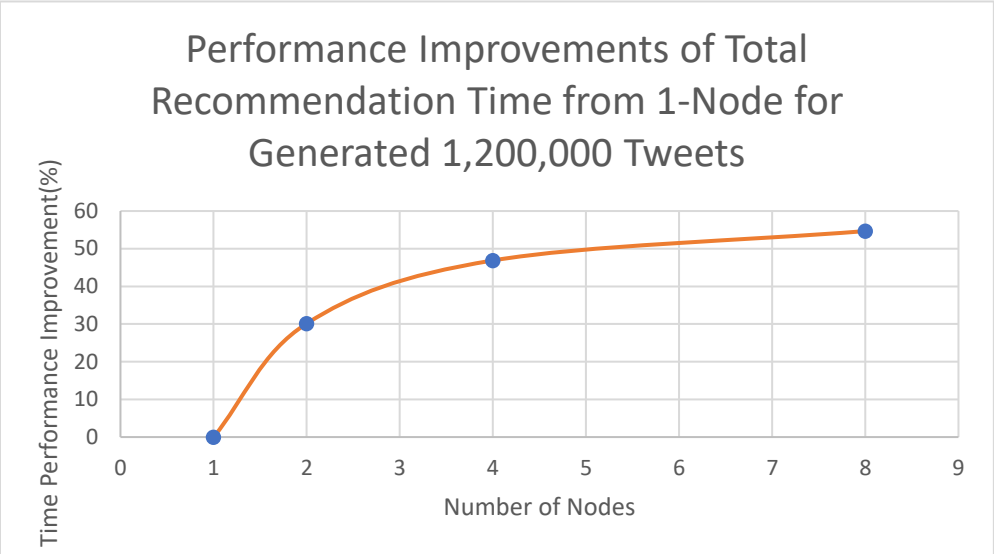

Fig. 16 Performance improvements of total recommendation time from 1 node for GEN_ALL424k_TO_1200k

As shown before, the performance improvements of GEN_ALL424k_TO_1200k reflect those in the smaller datasets. From the trends shown in these results, it was evident that performance improved as the number of nodes increased. The results from the artificial data followed the same trend as the real data. Thus, a recommender system running in parallel will perform better.

5 Conclusion and Future Work

5.1. Conclusion

The work in this thesis provides a method for generating a large amount of data for Twitter by using a Weibull distribution to generate *tf-idf* values to create words for a tweet. Scalability tests for a multi-agent system simulator of a Twitter recommender system are conducted. The performance improvements for using different configurations of 1, 2, 4, and 8 nodes against a different number of tweets were examined. The trends of the performance improvements for the recommendation algorithm, the communication costs, and the total recommendation time were similar between the real and artificial data. The simulator was validated and verified to ensure all the results were correct.

In the dataset for RU14k, the performance improvements of 56%, 89%, and 97% were found for 1 to 2 nodes, 1 to 4 nodes, and 1 to 8 nodes respectively. For the artifical dataset of GEN_RU14k, the performance improvements of 47%, 86%, and 99% were observed. The results of the artificial data were similar to the results of the real data. Examining the performance improvements of a larger dataset of ALL424k, it was shown to be 60%, 81% and 89%. The artificial dataset of GEN_ALL424k has the performance improvements of 51%, 79%, and 90%. In the larger datasets, the results of the artificial data and real data are still similar. From this observation, it can be concluded that the method for generating artificial data can generate data that is similar to the real data. As the datasets became larger, the results from the smaller dataset is closer to the results of the larger dataset (e.g. RU14k vs ALL424k, GEN_RU14k vs GEN_ALL424k). As such, when a larger dataset is provided, it can be assumed to have similar results to the smaller datasets. It can also be concluded that generating a larger dataset for scalability tests should provide results similar to the results that would be provided by a real dataset of the same size.

For the large amount of artificial data (i.e. over a million tweets) that was used for scalability test, the results showed that a clustering algorithm benefits more than the other algorithms of a recommender system when adding more nodes. For the clustering-based recommender system simulated in this work, just going from a central node to two or more nodes when distributing data evenly on nodes and running the clustering in parallel improves performance. The performance

improvements are approximately 60%, 80% and 90% for going from 1 to 2 nodes, 2 to 4 nodes, and 4 to 8 nodes respectively. The recommendations with multiple nodes were able to approximate the accuracy of a single node algorithm with the difference in MAE in the range of 2% to 6%, which is considered a good estimation. The overhead of communication cost because of the increasing number of nodes is marginal and it is not more than 250MB for eight nodes. From the results of the experiments, the performance improvement and communication costs show that the generated data is close to the real data from Twitter.

5.2. Limitations

The representation of the words in a tweet is a bag-of-words model, which means that the order of the words is not kept. This means the use of natural language processing on a sequence of words will not work with the generating artificial tweets method presented in this thesis. The recommendation algorithm used in the experiments is only one possible recommendation algorithm. Although the developed simulator has the capability of recommending follower to followee and vice versa based on similarity between users without clustering, the clustering method is used in the experiments because the goal of the conducted simulations is to test only the scalability of such a recommendation system when the accuracy can be measured in the simulated multi-node environment. The accuracy and performance improvement results obtained in these experiments may differ if another recommendation algorithm is used. The clustering-based recommendation algorithm used in the experiments has the quadratic time complexity which is subject to the scalability problem. The users in the datasets do not overlap in terms of who they were following when the data was collected i.e. the followers of each followee were not found to be following more than 1 followee in the group of followees that were collected, whereas it is possible for a user to have overlap on the followees when collecting data from Twitter. This case was not present in the data collected and not explored in the experiments.

5.2.1. Internal Validity Threats

The number of words in an artificial tweet were set to a pre-defined minimum and maximum value based on a dataset. The median was chosen as a maximum for computation simplicity. The distribution of the number of words in the tweets was not explored. The use of empirical distribution instead of Weibull distribution was not explored in these experiments since the simulation was determined to use a distribution with a given set of parameters.

5.2.2. External Validity Threats

The experiment conducted is only a simulation of a recommender system for Twitter that recommends a user on who to follow. There are different recommender systems built so the simplified model in the simulation experiments may not be the same as another recommender system in production. There were no physical components of each part of the systems to measure but the general parts of a system were simulated with the use of agents.

5.2.3. Curve Fitting Validity Threats

The curve fitting analysis used for generating artificial tweets suggested Weibull is the best model for the used datasets in this thesis. The curve fitting was done with the Expertfit software and the results in this thesis relied on its statistical tests. Only the visual observation was generated and used for comparison between the model and empirical distribution. The statistical tests for curve fitting for several cases of the used datasets suggested Gamma distribution is a better fit but the Weibull distribution was ranked as the best model for the majority of the datasets. In general, a best distribution model can be selected by finding the best model for the majority of the datasets when using the method described in the thesis. However, the best model may not be a Weibull distribution for other datasets collected from Twitter. In any case, the steps to generate the artificial tweets remain the same but a different distribution may be used. The best model is always empirical distribution, which can be used to generate random data but this requires storing the probabilities for every *tf-idf* value for each user. Using empirical distribution requires more storage space than the current method of using a distribution model with its given parameters. The choice was made to use a best distribution model with a given set of parameters since it is a general approach to generate random data for the purpose of simulation.

5.3. Future Work

The focus of this work was to test a common clustering method used in the recommendation algorithm, k-means clustering, with a list of similarity scores afterwards based on the clusters. It is worthy to look at the results for other types of clustering or algorithms used in recommender systems. Although the experiments provided promising results, there are still work that can be done in the future. There can be work done by looking at the topic similarity between the tweets of the users with the LDA model. The work done in these experiments only focused on

the k-means clustering algorithm with a list of similarity scores for the clusters for the recommendation algorithm. There are other algorithms for recommendation in literature that can be applied to check the scalability of the recommender system with the different configurations using the multi-agent system simulator. The links between the users and the behaviours or interests of a user were not explored in this work. In the future, the focus can be on the links between users based on the following relationship of Twitter. This can be explored by changing who a user agent is following during simulation and the effects that can be observed only through simulation.

References

[1] M. G. Armentano, D. Godoy, and A. Amandi, "Topology-Based Recommendation of Users in Micro-Blogging Communities," *Journal of Computer Science and Technology*, vol. 27, no. 3, pp. 624–634, 2012.

[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*, 2007, pp. 56–65.

[3] D. Bawden and L. Robinson, "The dark side of information: overload, anxiety and other paradoxes and pathologies," *Journal of Information Science*, vol. 35, no. 2, pp. 180–191, Apr. 2009.

[4] M. Nilashi, K. Bagherifard, O. Ibrahim, H. Alizadeh, L. Nojeem, and N. Roozegar, "Collaborative Filtering Recommender Systems," *Research Journal of Applied Sciences, Engineering and Technology*, vol. 5, no. 16, pp. 4169–4182, Apr. 2013.

[5] J. B. Schafer, D. Frankowski, J. Herlocker, and S. Sen, "Collaborative filtering recommender systems," in *The adaptive web*, Springer, 2007, pp. 291–324.

[6] M. Claypool, P. Le, M. Wased, and D. Brown, "Implicit interest indicators," in *Proceedings of the 6th international conference on Intelligent user interfaces*, Santa Fe, New Mexico, USA, 2001, pp. 33–40.

[7] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge & Data Engineering*, no. 6, pp. 734–749, 2005.

[8] C.-N. Ziegler, "Semantic web recommender systems," in *International Conference on Extending Database Technology*, 2004, pp. 78–89.

[9] J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*, 1998, pp. 43–52.

[10] N. N. Liu, X. Meng, C. Liu, and Q. Yang, "Wisdom of the better few: cold start recommendation via representative based rating elicitation," in *RecSys '11 Proceedings of the fifth ACM conference on Recommender systems*, Chicago, Illinois, USA, 2011, pp. 37–44.

[11] Y. Koren, R. Bell, and C. Volinsky, “Matrix Factorization Techniques For Recommender Systems,” *IEEE Transactions on Knowledge & Data Engineering*, vol. 42, no. 8, pp. 30–37, Aug. 2009.

[12] H.-F. Yu, C.-J. Hsieh, S. Si, and I. Dhillon, “Parallel matrix factorization for recommender systems,” *Knowledge and Information Systems*, vol. 41, no. 3, pp. 793–819, Dec. 2014.

[13] O. Chertov, A. Brun, A. Boyer, and M. Aleksandrova, “Comparative Analysis Of Neighborhood- Based Approache And Matrix Factorization In Recommender Systems,” *Eastern-European Journal of Enterprise Technologies*, vol. 3, pp. 4–9, 2015.

[14] M. Aleksandrova, A. Brun, A. Boyer, and O. Chertov, “Search for User-related Features in Matrix Factorization-based Recommender Systems,” in *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML/PKDD 2014), PhD Session Proceedings*, Nancy, France, 2014.

[15] M. Aleksandrova, A. Brun, A. Boyer, and O. Chertov, “Identifying representative users in matrix factorization-based recommender systems: application to solving the content-less new item cold-start problem,” *Journal of Intelligent Information Systems*, vol. 48, no. 2, pp. 365–397, Apr. 2017.

[16] Y. Yu and R. G. Qiu, “Followee Recommendation in Microblog Using Matrix Factorization Model with Structural Regularization,” *The Scientific World Journal*, vol. 2014, 2014.

[17] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[18] J. Li and A. Abhari, “Generating stochastic data to simulate a twitter user,” in *Proceedings of the 20th Communications & Networking Symposium*, Virginia Beach, VA, USA, 2017, pp. 102–112.

[19] A. M. Law and W. D. Kelton, *Simulation modeling and analysis*, 3rd ed. New York: McGraw-Hill, 2000.

[20] L. Breiman, *Statistics: with a view toward applications*. Boston: Houghton Mifflin, 1973.

[21] M. V. Menon, “Estimation of the Shape and Scale Parameters of the Weibull Distribution,” *Technometrics*, vol. 5, no. 2, pp. 175–182, 1963.

[22] D. R. Thoman, L. J. Bain, and C. E. Antle, “Inferences on the Parameters of the Weibull Distribution,” *Technometrics*, vol. 11, no. 3, pp. 445–460, 1969.

[23] M. Dastani, “Programming multi-agent systems,” *The Knowledge Engineering Review*, vol. 30, no. 4, pp. 394–418, 2015.

[24] Y. Shoham, “Agent-oriented programming,” *Artificial Intelligence*, vol. 60, pp. 51–92, 1993.

[25] V. K. Singh, N. Tiwari, and S. Garg, “Document Clustering Using K-Means, Heuristic K-Means and Fuzzy C-Means,” in *2011 International Conference on Computational Intelligence and Communication Networks*, Gwalior, India, 2011, pp. 297–301.

[26] A. Huang, “Similarity measures for text document clustering,” in *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, 2008, pp. 49–56.

[27] “Netlytic - social media text and social networks analyzer.” [Online]. Available: https://netlytic.org/. [Accessed: 26-Jul-2018].

[28] S. Y. Yang, A. Liu, and S. Y. K. Mo, “Twitter financial community modeling using agent based simulation,” in *Computational Intelligence for Financial Engineering & Economics (CIFEr), 2104 IEEE Conference on*, 2014, pp. 63–70.

[29] F. Hernández del Olmo and E. Gaudioso, “Evaluation of recommender systems: A new approach,” *Expert Systems with Applications*, vol. 35, no. 3, pp. 790–804, Oct. 2008.

[30] F. Meyer, F. Fessant, F. Clérot, and E. Gaussier, “Toward a New Protocol to Evaluate Recommender Systems,” in *Proceedings of the Workshop on Recommendation Utility Evaluation: Beyond RMSE*, Dublin, Ireland, 2012, vol. 910, pp. 9–14.

[31] G. Shani and A. Gunawardana, “Tutorial on application-oriented evaluation of recommendation systems,” *AI Communications*, vol. 26, no. 2, pp. 225–236, 2013.

[32] R. Saga, K. Okamoto, H. Tsuji, and K. Matsumoto, “Evaluating Recommender System Using Multiagent-Based Simulator,” *Recent Progress in Data Engineering and Internet Technology*, vol. 156, pp. 155–162, 2013.

[33] R. A. C. Capuruço and L. F. Capretz, “Evaluation and Assessment of Recommenders Using Monte Carlo Simulation,” in *Workshop and Poster Proceedings of the 20th Conference on User Modeling, Adaptation, and Personalization, CEUR Workshop*, Montreal, Quebec, Canada, 2012.

[34] H. Drachsler, H. Hummel, and R. Koper, “Using Simulations to Evaluate the Effects of Recommender Systems for Learners in Informal Learning Networks,” in *Proceedings of the*

2nd Workshop on Social Information Retrieval for Technology Enhanced Learning, Maastricht, The Netherlands, 2008, vol. 382, p. 15.

[35] Z. Cai, X. Guan, and Y. Li, "Collective Data-Sanitization for Preventing Sensitive Information Inference Attacks in Social Networks," *IEEE Transactions On Dependable And Secure Computing*, vol. 15, no. 4, pp. 577–590, 2018.

[36] Z. Cai, L. Perez, Z. Vegena, and C. Jermaine, "SimSQL," 2013. [Online]. Available: http://cmj4.web.rice.edu/SimSQL/SimSQL.html. [Accessed: 21-Jun-2018].

[37] N. Patki, R. Wedge, and K. Veeramachaneni, "The Synthetic Data Vault," in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Montreal, Quebec, Canada, 2016, pp. 399–410.

[38] D. Alvarez-Melis and M. Saveski, "Topic Modeling in Twitter: Aggregating Tweets by Conversations.," *ICWSM*, vol. 2016, pp. 519–522, 2016.

[39] L. Hong and B. D. Davison, "Empirical study of topic modeling in twitter," in *Proceedings of the first workshop on social media analytics*, Washington D.C., District of Columbia, 2010, pp. 80–88.

[40] K. Ryczko, A. Domurad, N. Buhagiar, and I. Tamblyn, "Hashkat: large-scale simulations of online social networks," *Social Network Analysis and Mining*, vol. 7, no. 1, pp. 1–13, Dec. 2017.

[41] N. Chomsky and M. Schützenberger, "The Algebraic Theory of Context-Free Languages," *Studies In Logic and the Foundation of Mathematics*, vol. 35, pp. 118–161, 1963.

[42] T. Bowden, "bowdens/tweet-simulator." [Online]. Available: https://github.com/bowdens/tweet-simulator. [Accessed: 24-Dec-2018].

[43] M. Woolf, "minimaxir/tweet-generator," *GitHub*. [Online]. Available: https://github.com/minimaxir/tweet-generator. [Accessed: 24-Dec-2018].

[44] A. M. Law, "How The Expertfit Distribution-Fitting Software Can Make Your Simulation Models More Valid," in *Proceedings of the 2011 Winter Simulation Conference*, Phoenix, Arizona, USA, 2011, pp. 63–69.

[45] E.-H. Han and G. Karypis, "Centroid-Based Document Classification: Analysis and Experimental Results," in *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*, London, UK, UK, 2000, pp. 424–431.

[46] R. C. Balabantaray, C. Sarma, and M. Jha, “Document clustering using K-means and K-medoids,” *International Journal of Knowledge Based Computer Systems*, vol. 1, no. 1, pp. 7–13, 2015.

[47] C. Buchta, M. Kober, I. Feinerer, and K. Hornik, “Spherical k-means clustering,” *Journal of Statistical Software*, vol. 50, no. 10, pp. 1–22, 2012.

[48] S. Al-Anazi, H. AlMahmoud, and I. Al-Turaiki, “Finding Similar Documents Using Different Clustering Techniques,” *Procedia Computer Science*, vol. 82, pp. 28–34, 2016.

[49] I. S. Dhillon and D. S. Modha, “Concept decompositions for large sparse text data using clustering,” *Machine learning*, vol. 42, no. 1–2, pp. 143–175, 2001.

[50] Y. Yamamoto, *Twitter4J - A Java library for the Twitter API*. Twitter4J.

[51] *Jade Site | Java Agent DEvelopment Framework*. Telecom Italia.

[52] J. Weng, E.-P. Lim, J. Jiang, and Q. He, “Twitterrank: finding topic-sensitive influential twitterers,” in *Proceedings of the third ACM international conference on Web search and data mining*, 2010, pp. 261–270.

[53] R. G. Sargent, “Verification and Validation of Simulation Models,” in *Proceedings of the 2011 Winter Simulation Conference*, 2011, pp. 183–198.

[54] I. S. Irobi, J. Andersson, and A. Wall, “Correctness criteria for models’ validation – A philosophical perspective,” in *Models, Simulations and Visualization conference (MSV’04)*, Las Vegas, NV, USA, 2004, pp. 1–8.

[55] M. A. Niazi, A. Hussain, and M. Kolberg, “Verification & validation of agent based simulations using the VOMAS (virtual overlay multi-agent system) approach,” in *Multi-Agent Logics, Languages, and Organisations Federated Workshops*, Torino, Italy, 2009.

[56] F. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Ltd, 2007.