MATRIX FACTORISATION BASED RECOMMENDATION FOR WEB MASHUPS

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF COMPUTER AND INFORMATION SCIENCES

Supervisor

Dr. Jian Yu

Dr. Sira Yongchareon

September 2020

By

Jacky Ou

School of Engineering, Computer and Mathematical Sciences

Abstract

In recommender systems, the Internet has evolved over the years for recommending items such as music, movies, books and videos for users to boost the popularity or sales for a single item. One of the significant challenges is the mashup developers spend much time to refine their searches to find suitable APIs (application programming interface). A framework is needed for incorporating the matrix factorisation (MF) recommendation that recommends APIs for a mashup application. In this project, we intend to build the recommender systems prototype by implementing machine learning to learn from the previous data extracted from the API description list. The contribution involves many processes of data collection, preprocessing, document vectorisation, implicit learning and recommendation. It should achieve the likelihood for a mashup application to invoke APIs provided by the data from ProgrammableWeb.

There are many MF algorithms available for use in recommendation systems. However, we find that many algorithms can suffer from data sparsity and cold-start issues. The recommendation approaches and filtering methods introduced in the literature review may provide ideas for conducting this investigation. We have employed evaluation metrics to investigate if the data fits well with the testing set. Finally, we highlight the limitations and possible future improvements to this study.

Contents

Abstract **2**

Attestation of Authorship **7**

Acknowledgements **8**

1 Introduction **9**
1.1 Motivation 9
1.2 Background 12
1.2.1 Recommender Systems 12
1.2.2 Collaborative Filtering 13
1.2.3 Matrix Factorisation 15
1.3 Research Question 17
1.4 Contribution 17
1.5 Thesis Structure 18

2 Literature Review **19**
2.1 Introduction 19
2.2 Filtering Methods 20
2.2.1 Collaborative Filtering 20
2.2.2 Content-Based Filtering 22
2.3 Matrix Factorisation 24
2.3.1 Probabilistic Matrix Factorisation 25
2.3.2 Non-Negative Matrix Factorisation 29
2.4 Recommendation Types 31
2.4.1 Service Recommendation 31
2.4.2 Social Recommendation 33
2.5 Conclusion 37

3 Research Method **39**
3.1 Introduction 39
3.2 Approach Overview 39
3.2.1 Data Preprocessing 41
3.2.2 Word Computation 43

3.3 Data Source . . . 45
3.4 Technical Details of the Proposed Method . . . 46
3.4.1 TF-IDF Calculation . . . 46
3.4.2 Cosine Similarity Calculation . . . 47
3.4.3 Model Learning and Increasing the Logarithm Function . . . 47
3.4.4 Loss Minimisation . . . 50
3.5 Conclusion . . . 50

4 Results and Discussion **52**
4.1 Introduction . . . 52
4.2 Experiments . . . 53
4.2.1 Experiment Setup . . . 53
4.2.2 Acceleration of the Performance . . . 53
4.2.3 Exporting Variables to CSV Files . . . 55
4.2.4 Evaluation Metrics . . . 55
4.2.5 How the Experiment was Conducted . . . 56
4.3 Word Extraction Results . . . 58
4.4 User-Item Matrix Results . . . 59
4.5 Experiment Results . . . 61
4.5.1 Logarithm Function Results . . . 61
4.5.2 Loss Function Results . . . 62
4.6 Evaluation Metrics Results . . . 63
4.6.1 MAE and RMSE Training Graph . . . 63
4.6.2 MAE and RMSE Training Results . . . 65
4.6.3 MAE and RMSE Testing Results . . . 65
4.7 Issues Encountered During Prototype Implementation . . . 66
4.7.1 Equation Flaws from the Journal Article . . . 66
4.7.2 Human Errors Lead to Unintentional Data Overfitting . . . 68
4.7.3 The Dilemma of Matrices A and M . . . 68
4.8 Loss Minimisation Training in Matrices A and M . . . 69
4.9 TF-IDF Word Vectorisation . . . 69
4.10 Conclusion . . . 70

5 Conclusion **71**
5.1 Introduction . . . 71
5.2 Answer to Research Questions . . . 71
5.3 Limitations . . . 72
5.4 Further Research . . . 74

References **82**

A Glossary and Abbreviations **83**

B Prototype Implementation **86**

List of Tables

2.1 Both matrices of the user-item ratings and friendship relations. 34

4.1 Top 40 words according to the document frequency (DF) values. . . . 58
4.2 The R^{prob} matrix after MF learning. 60
4.3 MAE and RMSE values in training and testing percentage ratios after training the loss value. 65
4.4 MAE and RMSE values evaluated from the testing data. 66

List of Figures

1.1 ProgrammableWeb. (n.d.). *Homepage of the ProgrammableWeb website* [Screenshot]. Retrieved from https://www.programmableweb.com 10
1.2 Example of the user, Bob searching for suitable APIs. 11

2.1 Gaussian distribution functions plotted in a graph. 25
2.2 PMF model. 26
2.3 PPMF model proposed by Gai (2014). 28
2.4 Diagram showing the proposed framework recommending suitable APIs (Fletcher, 2019). 33
2.5 The relationship between users and rated items. 35
2.6 Social circles with strong and weak ties. 36

3.1 Flowchart of overall processes. 40
3.2 Illustration showing the procedure of manual data preprocessing for manually omitting unnecessary APIs. Screenshot by author. 44
3.3 ProgrammableWeb. (n.d.). *Google Maps Engine API is stored in the ProgrammableWeb API directory* [Screenshot]. Retrieved from https://www.programmableweb.com/api/google-maps-engine 45
3.4 Diagram showing the Z_{ij} latent variable learning from S_{ij} and W_{ij}. . . 49
3.5 Diagram showing R_{ij} is learning from A_i and M_j. 51

4.1 Illustration showing the Anaconda prompt window. Screenshot by author. 54
4.2 Illustration showing the Jupyter Notebook user interface. Screenshot by author. 54
4.3 The example graph depicting the coordinate descent method (X. Wang, Zhang, Yan, Yuan & Zha, 2018). 57
4.4 Logarithm value over the number of iterations. 61
4.5 The graph depicting the loss value over the number of iterations. . . . 63
4.6 MAE training graph. 64
4.7 RMSE training graph. 64

Attestation of Authorship

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of student

Acknowledgements

First and foremost, I would like to express my gratitude to Dr Jian Yu for his guidance and support during my studies. Without him, this research would be impossible to continue. Secondly, I would like to thank Thi Thuy Mo Nguyen for sacrificing her time to evaluate my prototype and her suggestions on improving my programming. Finally, I would like to thank Dr Sira Yongchareon for clarifying my misunderstanding of statistical equations and machine learning.

Chapter 1

Introduction

1.1 Motivation

Let us recall in our daily activities where we utilise the Web to search for information, shop online, find friends via social network or watch video clips. Recommender systems provide a way to recommend items or services to the users without requiring explicit user inputs and tracks the data between users and items (Kunaver & Požrl, 2017). It can change how the users interact with the Internet, and the main goal for recommender systems is to increase sales when the users purchase a product that is interesting to their preferences (Shakirova, 2017). Recommender systems also a hot topic for research and was implemented in various items such as movies, TV shows, books, music and more (Ajoudanian & Abadeh, 2019).

In the era of web computing, not all items are rated because of the recent arrival of new products or a product has no user rankings. Any item that is not ranked by users can cause issues because no one has bought and used them before. To overcome the problem, one can utilise various methods to learn from the previous data and dynamically implement the results into each unrated item. Recommender systems use multiple algorithms such as data available in its database (e.g. ratings and social

relationships), the filtering algorithm used, techniques such as Bayesian networks and the quality of the results to predict the items for each user (Bobadilla, Ortega, Hernando & Gutiérrez, 2013). Many algorithms and techniques used for testing the recommendation accuracy can affect the results positively or negatively. Therefore, recommender systems provide an exciting topic for investigating how the items are recommended in the Web.

As the Web has improved over the years, the continuous development of new software and technologies has shaped the future of the World Wide Web and user interfaces. Also, as computers are evolving rapidly, users should be able to use the applications that are user-friendly and can achieve their tasks efficiently. In terms of mashup applications, the APIs (application programming interface) are used for the mashup developers to create services that rely on the other sources without needing to produce their programs for the Web.

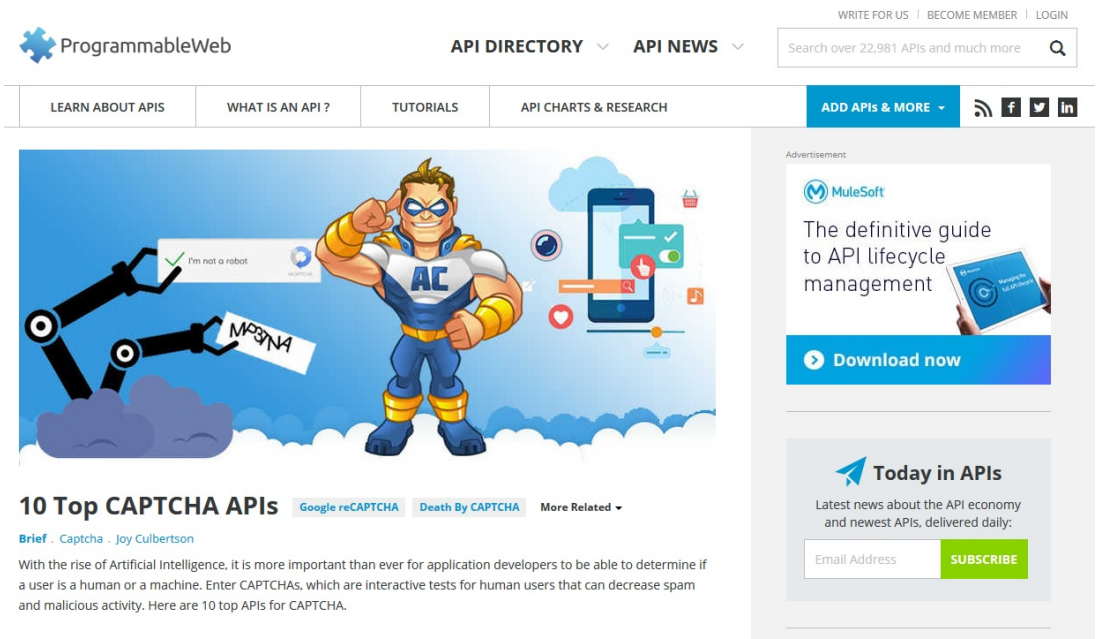

Figure 1.1: ProgrammableWeb. (n.d.). *Homepage of the ProgrammableWeb website* [Screenshot]. Retrieved from https://www.programmableweb.com

When a developer wants to create a mashup application, one can integrate several

APIs produced by other companies to achieve a specific function. The text query from the mashup application developer will be processed in the system to find the desired APIs (Zhong & Fan, 2017). For example, Bob wants to produce an application that will allow the user to check the package status with a map and a notification; however, his search results have returned nothing (B. Cao et al., 2017).

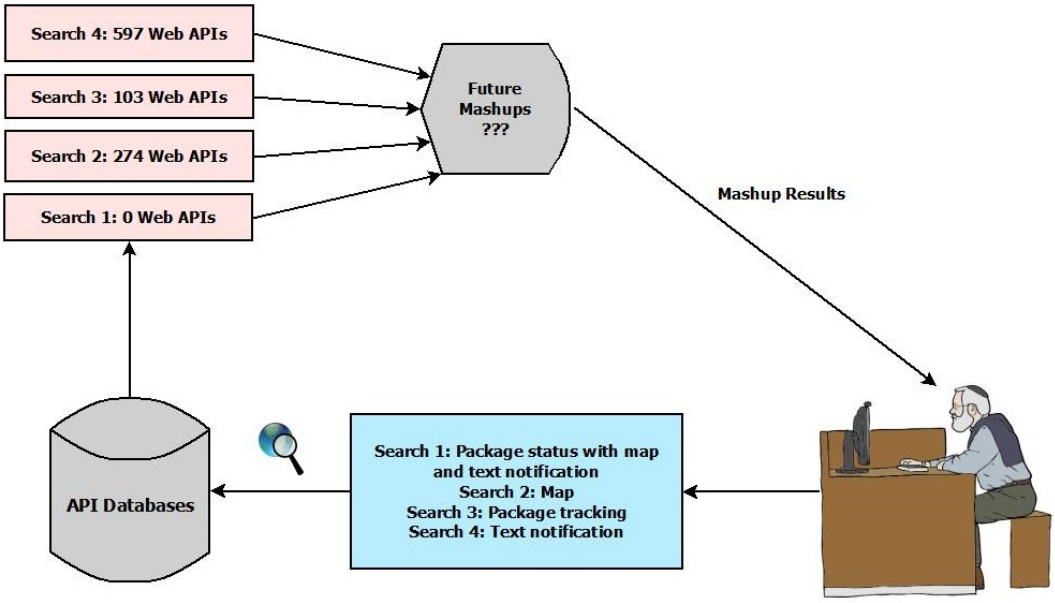

Figure 1.2: Example of the user, Bob searching for suitable APIs.

One problem for the situation is why did the ProgrammableWeb website search results display nothing? There are many reasons for returning none of the results. The common issue with selecting web APIs is a large number of APIs caused the search results to become vague, and as a result, it makes it difficult to choose correct APIs (B. Cao et al., 2017; Yao, Wang, Sheng, Benatallah & Huang, 2018). Another issue is that the search engine cannot handle a large number of keywords. Additionally, the application developers may remove the web APIs at any time without warning. There are various reasons for abandoning web APIs, including switching to a superior application or abandoning it permanently. As a result, it causes problems for mashup developers to replace the discontinued APIs. To alleviate the problem for developers, existing works

from Fletcher (2019), Hao, Fan, Tan and Zhang (2017), Cao et al. (2019) and Zhong and Fan (2017) could address the solutions for incompetent recommendations produced by search engines. However, the matrix factorisation (MF) based recommendation is the core subject of this research.

This thesis provides an insight into a recommendation system that recommends web APIs for a mashup application. Our aim for this study is to assist application developers to discover web APIs they require for its functionality, also to allow them to retain their trust in utilising search engines, and providing further assistance if necessary.

1.2 Background

1.2.1 Recommender Systems

Recommender systems have many methods to retrieve methods not just from the World Wide Web, but from many sources. The first mention of the term *recommender systems* happened in a journal article by Resnick and Varian (1997); however, this term is generic. Two filtering categories are introduced for recommender systems to address the generic term: Content-based filtering and collaborative filtering (CF).

Recommender systems have undertaken many improvements over the decades, and the research has contributed to enhancing the users' experience. As the history of recommender systems has shaped the future of computing, the related literature listed in the journal article by Sohrabi, Toloo, Moeini and Nalchigar (2015) described several authors have conducted the experiments and discussed the arguments whether the system is accurate or not. In the Scopus article list, there were more than 9400 articles and 76 reviews in the past, and it did not stop the growth of this topic (Calero Valdez, Ziefle & Verbert, 2016). The reason for the high interest of the research is the early mention of CF caused researchers to investigate the types and properties of the filtering

algorithms.

In 2006, the online media streaming service, Netflix announced a competition for anyone that can build recommender systems to predict movies and rewards $1 million to lower the root mean square error (RMSE) value by 10% (Amatriain & Basilico, 2016; Koren, Bell & Volinsky, 2009). Following the competition, at least one person won a prize for reducing the RMSE value by approximately 8%. With more time spent on improving the functionality of recommender systems, many algorithms were improving. In this day of age, recommender systems were not only used in web applications, and it also can be used for recommending consumable items in the Dota 2 video game (Looi, Dhaliwal, Alhajj & Rokne, 2019). In another case that recommender systems were used for exchanging books without needing to utilise the wish list (Pera & Ng, 2018). As deep learning and artificial intelligence (AI) are continually improving, the computing systems have improved the CF techniques for recommending new items.

1.2.2 Collaborative Filtering

CF has been around in many years for recommending specific items for users with their preferences. The first mention of CF is Goldberg, Nichols, Oki and Terry (1992), where they utilised filtering on emails, and an illustration showed most information filtering techniques could overwhelm an employee's mindset. Another mention is Resnick, Iacovou, Suchak, Bergstrom and Riedl (1994) discussed different CF techniques that allow users to filter out uninterested news by using the GROUPLENS architecture. As the early CF algorithms developed to alleviate the problem of unfiltered or unused items, the CF method should provide insights into shaping the future of consumer interactions.

With the development of CF methods, both works from Goldberg et al. (1992) and Resnick et al. (1994) shaped the future of recommender systems. Since they have completed their experiment on the email and news filtering mechanisms, the CF has

evolved to extend its use on other applications. As a result, computing technologies have evolved quickly, and the most significant milestone is that the World Wide Web has caught the attention of the public eye. Since the users of the World Wide Web grew, the online retailer, Amazon recommended various items to the customers and has been around for more than two decades. Additionally, Amazon has implemented the item-based CF algorithm in 1998 and the user-based CF framework in 2003 (Smith & Linden, 2017). In the online shopping environment, new items were constantly added to the database.

In the 2010s, CF is continuously evolving, and numerous research papers proposed new prototypes that combine itself with other algorithms. There were many cases that CF was utilised in many situations. In respect to combining content-based filtering and CF methods altogether, Zhao, Yan, Jin, Yang and Yu (2017) conducted the experiment on utilising both methods for mobile searching based on user's behaviour such as shopping, social network and entertainment. In music recommendation, Iwahama, Hijikata and Nishida (2004) proposed a prototype that recommends music based on user ratings and song characteristics such as tone and pitch. Dong and Shen (2013) used the same hybrid approach for recommending microblogs by collecting data from Sina Weibo. Finally, Yao, Sheng, Segev and Yu (2013) tested a hybrid framework that integrates CF and content-based filtering approach for combining data with the newly added user preferences. Although Xiong, Wang, Zhang and Ma (2018) compared their work to Yao et al. (2013) for the hybrid approach content-based recommendation, they proposed a prototype that includes a neural network approach for hybrid CF algorithms. Not only CF can cooperate with content-based filtering, but this may also be possible for developing the prototype for this research.

Multiple papers are documenting the combination of unrelated hybrid algorithms with CF. To give an example, Jain, Liu and Yu (2015) utilised MF that utilises the CF method of a user-item matrix to predict and create mashups. They have used the

user-item matrix assuming the API denotes the user, and the mashup represents the item. The MF combined with CF method is the main topic of focus in this study. As authors have combined CF with other methods, it still needs more work to improve the recommendation results further.

Although the CF recommendation is always reforming, many journal authors have used CF to test the algorithm for use in case studies. For example, Ajoudanian and Abadeh (2019) have proposed a recommendation system that recommends human resources to GitHub project leaders. Another example is Zhang, Meng, Kong and Dong (2019) utilised CF to recommend electricity plan based on single-rate and time-of-use tariffs. For movies, Pal, Parhi and Aggarwal (2017) conducted the experiment to test the performance of the mean absolute error (MAE) values for recommending movies by comparing Hybrid CF, Pure CF and SVD (singular value decomposition) methods. Not just CF is used in online shopping, music and videos; books are also popular for bibliophiles. Tewari, Kumar and Barman (2014) presented a new approach for combining CF and content-based filtering that recommends books based on the book information, category and order information. In medical fields, Zang, An and Hu (2014) conducted the experiment that utilises CF to recommend healthy living programs for patients suffering chronic diseases based on the information such as blood pressure and body mass index. Many authors mentioned in case studies and experiments have demonstrated their efforts in promoting CF in the recommender systems research.

1.2.3 Matrix Factorisation

MF was known for its method to solve linear algebra equations (Hubert, Meulman & Heiser, 2000). It uses the matrix multiplication method to calculate both matrices by multiplying all rows by each column, similar to the dot product computation. Additionally, MF models implicitly learn from both existing users and items matrices which can

generate new predictions (Shi, Larson & Hanjalic, 2014). The earliest mention of MF could be around the late 1990s; however, Hubert et al. (2000) introduced the idea of utilising this method to solve some issues related to various science-related fields. However, MF is a generic term for using linear algebra to multiply both matrices together. The algorithms that utilise the MF architecture are probabilistic matrix factorisation (PMF), non-negative matrix factorisation (NMF) and singular value decomposition (SVD).

MF is essential for this study because we will implement it in our experiment. Many authors have documented the use of MF in testing the function. One example of the MF utilisation is G. Wang, Jiang, Wang and Yang (2019) proposed the idea of the PMF prototype that recommends researchers and postgraduates to the list of groups based on the information from scholarly articles they interact. That approach could increase the potentiality of the growth of groups that shares the idea of academic research. Another example is Zetlaoui, Feinberg, Verger and Clémençon (2011) tested the NMF model for predicting the assessment of the food choices for the adult population. Their study could assist many people around the world in being educated by their food choices. Nonetheless, the authors have listed the limitations of their research, and the sparsity of the food data may affect the accuracy of the prediction results.

There are many uses of MF, and in most cases, various authors have proposed new subvariants of MF and not just limited to PMF and NMF. The reason for multiple authors attempting to create a new model under the existing MF methods is to test its reliability in comparison to the standard models. In this case, many papers were published to showcase their studies for integrating other models. Jiang, Li and Xu (2019) proposed a new NMF algorithm called relative pairwise relationship - non-negative matrix factorisation (RPR-NMF) that uses the pairwise relationship model to increase the recommendation accuracy. Another example of NMF variant is S. Zhang, Wang, Ford and Makedon (2006) modified the NMF algorithm to create weighted non-negative matrix factorisation (WNMF) to guarantee the shortened time to reach its convergence

point. Kim, Park, Oh and Yu (2017) integrated the convolutional neural networks (CNN) with the PMF model. Their idea of CNN is not relevant to this study; nonetheless, it could be useful for future studies of recommender systems.

1.3 Research Question

The research questions of this thesis are:

1. **How do we integrate context information such as title, description and tags from APIs in order to fill in the unrelated API-mashup relationships?**
   The dataset information from API titles, descriptions and tags are sufficient for learning the missing spots of the API-mashup matrix. We could use data preprocessing to manipulate information from API lists and adopt the TF-IDF (term frequency - inverse document frequency) in order to vectorise the set of numbers.

2. **How do existing API-mashup invocations affect the accuracy of the recommendation results?**
   A reliable recommendation method should produce consistent results. We could utilise machine learning to allow existing invocations to determine the relationship strength between APIs and mashups.

1.4 Contribution

The contribution of this thesis is an MF based recommender systems prototype for recommending web APIs. We will extract contents from the API descriptions and utilise the machine learning method to recommend APIs for every mashup. Five parts separate the experiment process: 1) Retrieval of API and mashup information from

ProgrammableWeb; 2) Performing data preprocessing of the API dataset; 3) Corpora to vectorised documents conversion; 4) Utilisation of machine learning to fill in the new relationship between unrelated APIs and mashups; 5) Recommendation following the MF algorithm.

Besides, this thesis also discusses different types of recommendation algorithms related to MF, filtering techniques and recommendation types used to recommend items or services. The results from recommendation experiment will be presented and analysed.

1.5 Thesis Structure

In this chapter, we have introduced the background of mashup creation, recommender systems, CF and MF. The questions were defined in this research.

In the second chapter, we introduce a literature review. To begin, we introduce filtering methods. We discuss the MF techniques and their variants. Finally, we provide a background to the recommendation techniques used in recommender systems.

In the third chapter, we discuss the research methodology and the procedures of preprocessing the data. The equations for this experiment will be provided based on the article from Yao et al. (2018) to understand the function of the program.

In the fourth chapter, we report and discuss the results from the experiment and describe the flaws of the paper. Additionally, we will explain how we overcome issues for the experiment.

In the fifth chapter, we conclude this thesis. We discuss the limitations of this research and suggest improvements for future works.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, we explore the different methods of recommender systems to understand how different recommendation algorithms work. We will conduct a literature review relating to the relevant topics of recommender systems and its indirect connections to this topic of research. Conducting a literature review not only requires us to read various papers and summarise past works but also, we need to link their ideas into this study (Punyabukkana, 2017).

First, we elaborate the filtering methods in this literature review and introduce their variants. We explain the collaborative filtering (CF) topic and illustrate the similarity calculations to describe the main points of this filtering method. Second, we introduce recommendation approaches based on this study. Third, we explain the recommendation approaches to provide a fundamental understanding of different ways of recommendation. Finally, we describe the matrix factorisation (MF) methods in this chapter, and this is important for reflecting other author's previous works into this research.

2.2 Filtering Methods

2.2.1 Collaborative Filtering

The purpose of CF is to recommend an item based on the data from the user's rating for each item. For this type of filtering method, it will be in the form of an $m \times n$ matrix, where m is the rows of users and n denotes the columns of items (Schein, Popescul, Ungar & Pennock, 2002). As users provided sufficient information to rate an item, the system can recommend an item for another user based on the submitted ratings and the most in common groups (Bobadilla et al., 2013; Cohen, Aharon, Koren, Somekh & Nissim, 2017). While CF can produce consistent results, the downside is that new users and data sparsity issues can suffer prediction issues and can cause the prediction of items to halt (Turnip, Nurjanah & Kusumo, 2017). Contrary to the disadvantage of CF, it undertook many changes to increase the accuracy of predictions in many ways. With many journal articles and conference papers were published continuously, many authors have documented their new variants of CF recommendation algorithms. However, there are many challenges of the CF algorithms, and many questions should be asked, for example, prediction, rank and classification accuracy; how evaluation should be executed and selecting the suitable algorithm for performing this task (Cacheda, Carneiro, Fernández & Formoso, 2011). We will introduce the existing types of CF methods and describe the works that various authors of multiple journals had done.

The memory-based CF utilises the whole or a sample of the preference database to predict suggested items for the user based on their past preferences or the neighbours of the same interest (Kim & Ahn, 2011; Shakirova, 2017; Su & Khoshgoftaar, 2009; Xie et al., 2007). The standard technique used for the memory-based CF is the *k*-nearest neighbours (*k*NN) algorithm. Its purpose is to select the closest defined number of neighbours for predictions which the k users with the most extraordinary likeness are

chosen (Shi et al., 2014). Moreover, the three tasks of the *k*NN algorithm are (Bobadilla et al., 2013): (1) Select the k number of neighbours; (2) predict the ratings by utilising the aggregation approach; (3) retrieve the results and select the top N recommendations. The disadvantage of the *k*NN algorithm is that if the data is sparse, the algorithm will produce unwanted results. Because of the sparsity, the users may not have much time to rate all items. Another drawback is that the low scalability issue can decrease the recommendation performance due to a large number of users and items (Bobadilla et al., 2013). As a result, the kNN algorithm may not be stable for predicting a large amount of sparse data, although it will work well on massive data.

The neighbourhood-based CF uses the similarity calculation and the collection of the mean ratings of all or specific users or items (Feng, Liang, Song & Wang, 2020; Yao, Sheng, Ngu, Yu & Segev, 2015). Additionally, it uses various methods to calculate the similarity values based on the user-item, item-based or user-based ratings. The item-based CF method is the most common CF approach for recommender systems (Verma, Mittal & Agarwal, 2013). The correlation-based similarity equations are always used for computing the similarity between users or items. The Pearson correlation equation for the similarity computation is mentioned in many articles similar to this topic (Su & Khoshgoftaar, 2009). The equation for the correlation-based similarity is

$$w_{uv} = \frac{\sum_{i \in I}(r_{ui} - r_u^{average})(r_{vi} - r_v^{average})}{\sqrt{\sum_{i \in I}(r_{ui} - r_u^{average})^2}\sqrt{\sum_{i \in I}(r_{vi} - r_v^{average})^2}} \tag{2.1}$$

where u and v are users denoted as $u = \{1, 2, ..., n\}$ and $v = \{1, 2, ..., n\}$, $i \in I$ are summations that items were rated by users u and v, r is the actual rating from users, and $r^{average}$ is the average rating. The equation from neighbourhood-based CF is the fundamental block for the filtering algorithm, and it is thought to compensate for different ratings from various users and items which can cause bias (Hu, Koren & Volinsky, 2008). Another method to calculate the similarity between two items or users

is the vector-based cosine similarity. The equation for the vector-based cosine similarity is

$$s_{ij} = \frac{Vector(i) \cdot Vector(j)}{||Vector(i)|| \times ||Vector(j)||} = \frac{i_1 i_2 + j_1 j_2}{\sqrt{i_1^2 + j_1^2}\sqrt{i_2^2 + j_2^2}} \tag{2.2}$$

The difference between Equations 2.1 and 2.2 is the cosine similarity only requires two vector variables which the numerator calculates the dot product and the numerator calculates the norm. Likewise, the correlation-based similarity requires variables from actual and average ratings. Therefore, if similarity computations are mathematically correct, the recommendation system can produce results in common when two users have similar tastes (H. Wang, Tao, Yu, Lin & Hong, 2018).

Despite the existing CF techniques introduced in this literature review, there are many variants of CF created mostly with combining other recommendation techniques. As a result, it becomes a hybrid approach for CF. Many articles have documented the creation of the hybrid CF systems that the main goal is to test the effectiveness of the algorithms. One good example is Yao et al. (2013) integrated the CF and the content-based filtering recommendation to create a new recommendation model by combining the web services, its descriptors and the users. They have proposed a hybrid approach of CF, developed the prototype and compared the results against standard CF and content-based filtering recommendation.

As CF is most commonly used for recommending items for users, they are the most crucial aspect of the topic for recommender systems. Despite many variations of this function, CF is considered the most favourite topic of the recommender systems research.

2.2.2 Content-Based Filtering

Content-based filtering was used for recommending items based on the user past behaviours (Bobadilla et al., 2013). Unlike CF, the content-based filtering does not

utilise the user-item rating data to predict recommendations (Su & Khoshgoftaar, 2009). The vital aspect of content-based filtering method is the aggregation of the user's metadata that the filtering system can retrieve the information from his/her interests (Bogers & Van den Bosch, 2009). For the perspective of an item, e.g. a film has titles, actors, directors and the publisher, which all of them can be used for content-based filtering (Salter & Antonopoulos, 2006). Consequently, that leads to recommending items based on the user's or item's metadata.

To elaborate on the early history of content-based filtering, Goldberg et al. (1992) experimented with filtering electronic documents for the office worker to reduce anxieties. The overflow of the incoming documents can frustrate an office worker because the unfiltered items made it overwhelming to read them all. The authors in the Tapestry journal article proposed a prototype that utilises the client-server architecture to filter out unnecessary documents, including emails that provide ineffectual information for an office worker. Another one is Resnick et al. (1994) proposed a GROUPLENS architecture for filtering the news articles by the Netnews platform. It provides similar recommendation systems that use the rating matrix to give better results. The results for the experiment is the authors of the Netnews journal had a successful experiment that uses the subgroups to provide a recommendation for the news articles.

Content-based filtering is equally essential with CF because, in the perspective of the user, the website's recommender systems need to recommend items for the user who have used the item from the same category. The content-based filtering method often works together with CF, and the integration of both methods can enhance the results of the recommendation (Adomavicius & Tuzhilin, 2005). For the hybrid filtering algorithm to work, Basilico and Hofmann (2004) combined the kernels from the variables of identity, attribute, correlation and the quadratic correlation kernels. This method of combining both filtering algorithms is a hybrid approach for allowing the two to work together. Despite that combining kernels is a way for utilising a hybrid CF

recommendation, one can simply use metadata of an item and calculate the similarity values of vectorised corpora.

2.3 Matrix Factorisation

MF is a standard method to provide implicit feedback to any items that are not explicitly ranked by users and is said to produce high accuracy predictions (G. Wang et al., 2019; Zhang, Liu, Chun-Gui, Wei & Huiyi-Ma, 2014). It symbolises the users and items where it represents the vectors from the item rating patterns (Koren et al., 2009). MF does implement machine learning to train the non-ranked items by utilising the training data. The past publications regarding recommending TV shows by using MF has been implemented in the experiment regardless of the inattention it receives in the research literature (Yang, Guo, Liu & Steck, 2014). As a result, several past works of literature should influence MF to receive more attention in respect to developing its ongoing research potential. The main goal of MF is to solve the latent factors from users and items based on the direct relationship between the two, where the user has interacted with the item (Kim et al., 2017). In terms of recommending APIs (application programming interface) for a mashup application, the system should learn from the existing API co-invocations or API-mashup invocations. Most API pairs may have little or no relations, and it is problematic. However, if at least one user used an unranked item and gave a detailed review, the actual rating can replace an implicit one. The disadvantage of MF methods is that the resulting model becomes static following the computation of the matrices (Aghdam, Analoui & Kabiri, 2016). Under those circumstances, recommender systems must update continuously to overcome the static model issues.

In each subsection, we will also discuss one subvariant of the algorithms that was proposed by the authors of a journal article. The discussion will illustrate the minor

differences in the slightly modified algorithms discussed in different literature.

2.3.1 Probabilistic Matrix Factorisation

In probabilistic matrix factorisation (PMF), it is based on a Gaussian distribution model. The Gaussian distribution is used for calculating the probability based on the mean value of the function. The general formula of the Gaussian distribution is denoted as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-u)^2}{2\sigma^2}} \tag{2.3}$$

where σ is the standard deviation, σ^2 is the variance, x is the x-axis value, and u is the mean value for the equation. If a Gaussian distribution equation is plotted on the graph, it should look like a bell-shaped curve in Figure 2.1. The standard deviation of

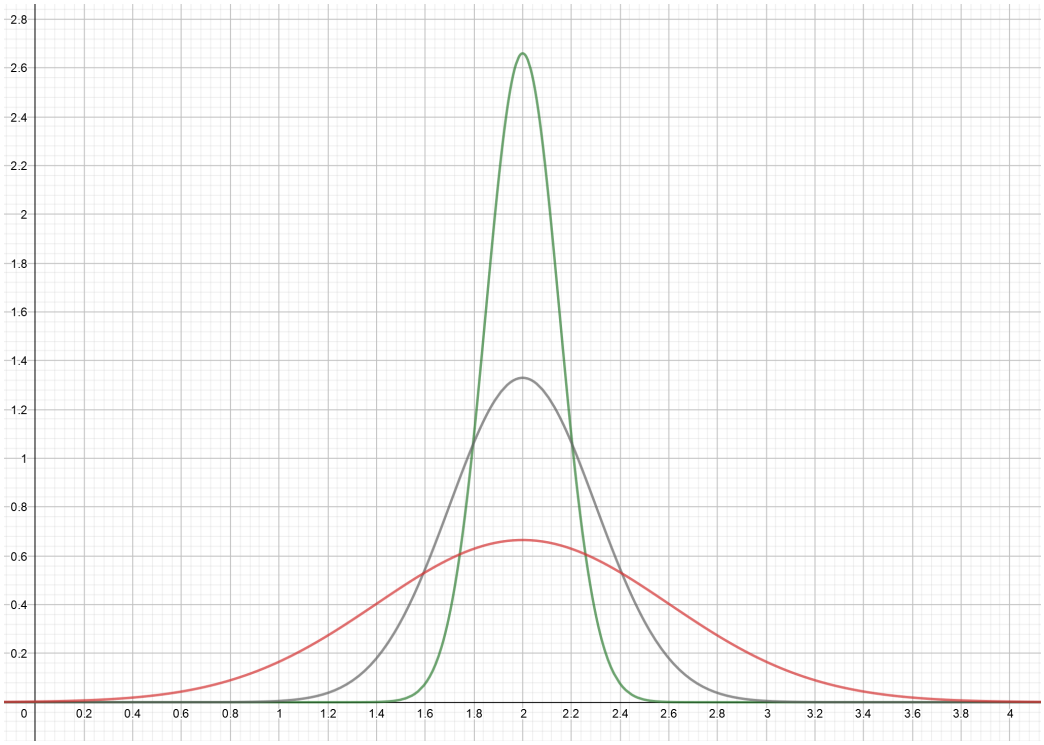

Figure 2.1: Gaussian distribution functions plotted in a graph.

Gaussian distribution graphs control the spread of the data as it increases or decreases

the interquartile length of the graph. If the standard deviation value is large, it increases the size of the difference between quartiles (x-value) and decreases the y-value. In another scenario, if the standard deviation is small, it does the opposite.

Any variables that have unknown values should be assigned with random variables based on the Gaussian distribution values. To have the easy reference on the Gaussian distribution expression, the normal distribution notation $N(\mu, \sigma^2)$ is used.

PMF is used for calculating the probability by using the Gaussian distribution. Let R_{ij} be the $n \times m$ matrix where it stores the probability of users invoking the items where $R_{ij} \in \{x \mid 0 \leq x \leq 1 \mid x \in \mathbf{R}\}$. Figure 2.2 shows that R_{ij} is learned from U_i and V_j,

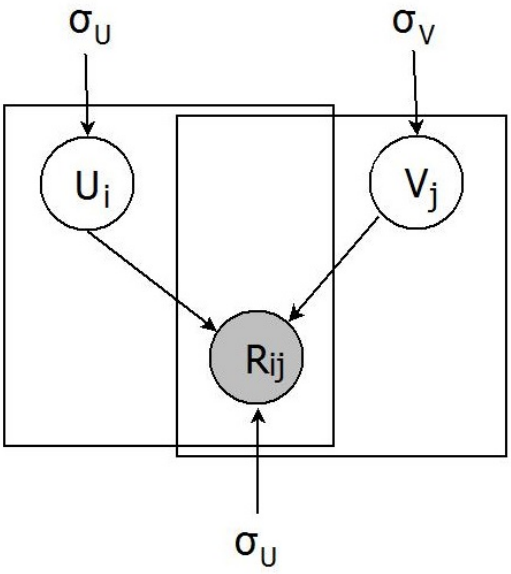

Figure 2.2: PMF model.

where i is the user row number of the matrix U as $i = \{1, 2, ..., n\}$, and j is the item row number of the matrix V as $j = \{1, 2, ..., n\}$. The standard deviations are applied to all variables to ensure the suitable spread of the probability distribution. The variables used for PMF should be assigned a numeric value based on a Gaussian distribution. The

Gaussian distribution for PMF is defined as (Gai, 2014)

$$p(R|U,V,\sigma^2) = \prod_{i=1}^{M}\prod_{j=1}^{N}[N(R_{ij}|U_iV_j^T,\sigma^2)]^{h((u,i)\epsilon D)} \tag{2.4}$$

where i is the row index of a matrix denoted as $i = \{1,2,...,n\}$ with the maximum M, and j is the column index as $j = \{1,2,...,n\}$ with the maximum N. In later equations with the same notations for referencing rows and columns in a matrix, we do not have to repeat the statement. To reduce overfitting of the PMF model, the zero mean spherical vector should be assigned on variables U and V (Y. Cao, Li & Zheng, 2019)

$$p(U|\sigma_U^2) = \prod_{i=1}^{M}[N(U_i|0,\sigma_U^2 I_K)] \tag{2.5}$$

$$p(V|\sigma_V^2) = \prod_{j=1}^{N}[N(V_j|0,\sigma_V^2 I_K)] \tag{2.6}$$

where $h(x) = 1$ if x is true, and I_K is a K-dimensional identity matrix.

The general equation of the PMF loss function is used where the final value is minimised by learning the latent variables U and V (Mnih & Salakhutdinov, 2008)

$$\min(U,V) = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N} I_{ij}(R_{ij} - U_i^T V_j)^2 + \frac{\lambda_U}{2}\sum_{i=1}^{M}||U_i||_F^2 + \frac{\lambda_V}{2}\sum_{j=1}^{N}||V_j||_F^2 \tag{2.7}$$

The main variables in the loss function are U and V, where U denotes the users and V is the items. Additionally, the sum of the Frobenius norm in both U and V are appended to reduce the bias.

Pairwise Probabilistic Matrix Factorisation

One variant of the PMF method should be described in detail. The PMF can be slightly modified to include more features for testing the pairwise model effectiveness in MF. Gai (2014) proposed a slightly modified PMF algorithm called pairwise probabilistic

matrix factorisation (PPMF). PPMF is a variant of the PMF framework that includes another variable for pairwise operations. The PPMF model is most similar to PMF.

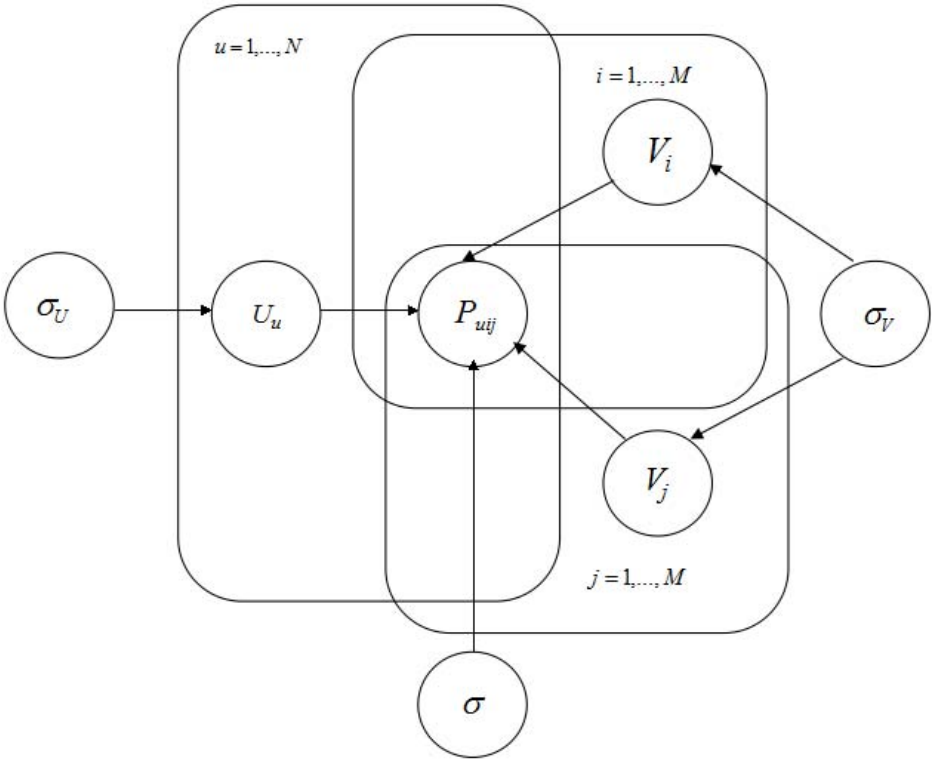

Figure 2.3: PPMF model proposed by Gai (2014).

One can notice that V_i has a repeated variable of V_j because of pairwise operations for the PPMF algorithm. If V_j is omitted, it becomes a standard PMF model. The general equation of PPMF looked similar to the PMF function. Also, the equation can be modified to suit the nature of minimising the loss function based on the three main variables utilised to learn P_{uij}. The P_{uij} variable will become the tensor matrix because of the three-dimensional shape in a whole framework. The distribution function for the PPMF model will become

$$p(P|U,V,\sigma^2) = \prod_{u=1}^{M}\prod_{i=1}^{N}\prod_{j=1}^{N}[N(P_{uij}|g(U_u V_i^T - U_u V_j^T),\sigma^2)]^{h((u,i,j)\epsilon T)} \tag{2.8}$$

$$p(U_u|\sigma_U^2) = N(U_u|0,\sigma_U^2 I) \tag{2.9}$$

$$p(V_i|\sigma_V^2) = N(V_i|0, \sigma_V^2 I) \tag{2.10}$$

$$p(V_j|\sigma_V^2) = N(V_j|0, \sigma_V^2 I) \tag{2.11}$$

and the cost function should be

$$min(U, V_i, V_j) = \frac{1}{2}\sum_{u=1}^{M}\sum_{i=1}^{N}\sum_{j=1}^{N}(P_{uij} - g(U_u V_i^T - U_u V_j^T))^{2\times h((u,i,j)\epsilon T)} + \frac{\lambda_U}{2}\sum_{u=1}^{M}||U_u||^2 + \frac{\lambda_V}{2}\sum_{i=1}^{N}||V_i||^2 + \frac{\lambda_V}{2}\sum_{j=1}^{N}||V_j||^2 \tag{2.12}$$

Notice that the cost function and the conditional distribution function has three variables needed to learn the latent variable of R_{uij}. Nevertheless, the PPMF model relationship diagram would be similar to a standard PMF relationship diagram.

2.3.2 Non-Negative Matrix Factorisation

Non-negative matrix factorisation (NMF) is commonly investigated as it is the most popular method to determine the ratings of an item. The difference between NMF and other MF techniques is that NMF does not allow any negative values in a matrix. Despite the requirement for positive values, a few negative values in every matrix does not affect the output of the result (Fogel, Hawkins, Beecher, Luta & Young, 2013). Otherwise, any matrix that has many negative values does not satisfy the properties of NMF. The drawback of NMF is that it may be possible to contain null values as the zero values are allowed in the matrix (Zetlaoui et al., 2011). The null values in the matrices may pose issues with MF procedures to calculate the multiplication of two matrices. The general equation for NMF defined by Lee and Seung (2001) is

$$min(U, V) = \sum_{i=1}^{M}\sum_{j=1}^{N}(U_{ij}log(\frac{U_{ij}}{V_{ij}}) - U_{ij} + V_{ij}) \tag{2.13}$$

Notice that U_{ij} is the numerator over V_{ij} in the argument of the logarithm operation. If any negative values are present in the matrix, it can output null or invalid values from the logarithm operation. To overcome the issue, the refined equation for the NMF loss function that can accommodate a few negative values as defined by Aghdam et al. (2016) is

$$min(U,V) = \omega(\sum_{i=1}^{M}\sum_{j=1}^{N}(R_{ij}-\sum_{f=1}^{K}(U_{if}V_{fj}^{T}))^{2})+\phi(\sum_{i=1}^{M}\sum_{j=1}^{N}(R_{ij}ln(\frac{R_{ij}}{[UV^{T}]_{ij}})-R_{ij}+[UV^{T}]_{ij})) \tag{2.14}$$

where ω and ϕ dictate the divergence and the Frobenius norm. The defined values of the learning parameters are $\omega \in \{x \mid 0 \leq x \leq 1 \mid x \in \mathbf{R}\}$ and $\phi = 1 - \omega$. In this function, the variables U denotes users, and I denotes items where $U \geq 0$ and $V \geq 0$. Note that UV^{T} is the standard matrix multiplication for both items U and V. Both variables allow minor amounts of negative values. Thus, the NMF cost function proposed by Aghdam et al. (2016) reduces the incidence of encountering the negative value argument error in the logarithm function. In contrast, Equation 2.13 cannot tolerate any elements with negative numbers in both matrices U and V.

Weighted Non-Negative Matrix Factorisation

There is a subvariant of the NMF method, and it is called the weighted non-negative matrix factorisation (WNMF). The work of S. Zhang et al. (2006) introduced the WNMF as the alternative to NMF, and it is said to guarantee for reaching its convergence. Nonetheless, it is not determined to have optimum values following training. The logarithm equation for WNMF is

$$log(A,X) = -\frac{1}{2\sigma^{2}}\sum_{i=1}^{M}\sum_{j=1}^{N}(A_{ij}-X_{ij})^{2} + C \tag{2.15}$$

where A_{ij} is the rating matrix with some missing entries, X_{ij} is the linear model matrix, and C is the constant for the function. The constant of this function is optional and can be substituted by various functions in order to adjust the results if necessary. If matrix A_{ij} has the unknown values, it will be randomly assigned by the Gaussian distribution of $N(X_{ij}, \sigma^2)$. The expectation-maximisation procedure is (S. Zhang et al., 2006)

$$log(A, X) = -\frac{1}{2\sigma^2}(\sum_{A_{ij} \epsilon A_O} (A_{ij} - X_{ij})^2 + \sum_{A_{ij} \epsilon A_U} (X_{ij}^{(t-1)} - X_{ij})^2) + C \tag{2.16}$$

where A_O is the observed A rating matrix and A_U is the unknown A matrix. S. Zhang et al. (2006) results concluded that the WNMF approach performed faster than the expectation-maximisation procedure. To prove the author's statement is not contradictory, note the additional summation equation after the first summation used in the WNMF approach. Computing the difference between $X_{ij}^{(t-1)}$ and X_{ij} may cause the program to slow down or encounter the array out-of-bound errors that will throw an exception. However, with careful implementation, the program can work flawlessly.

2.4 Recommendation Types

2.4.1 Service Recommendation

Service recommendation is frequently used for recommending specific web items for the users. An essential aspect of service recommendation is the process of user interactions and its algorithm for predicting the correct services. In this study, web APIs are part of the web services. The requirements for the web service recommendation should have, according to Yao et al. (2015): High recommendation accuracy, recommendation serendipity and reliable recommendation of newly deployed services. There are many ways that the system can recommend APIs for the new proposed mashups. The interactions used for recommending API services are: Direct user-to-web API interaction,

direct user-to-mashup interaction and indirect user-to-user interaction (Fletcher, 2019). The direct user-to-web API interaction method may provide a comprehensive approach for recommending API services. Also, the direct user-to-mashup interaction method offers the same way as the user-to-web API interaction procedure. The only drawback in the direct user-to-item interactions is that if the data is sparse, it may not produce the best results (Mu, Xiao, Tang, Luo & Yin, 2019). Indirect user-to-user interaction can follow the social recommendation approach for recommending APIs. However, if there is insufficient data for users interaction with APIs or mashups, the results produced are inferior, thus garbage in, garbage out.

To begin with service recommendation, several authors (Cao et al., 2019; Hao et al., 2017; Yao et al., 2018; Zhong & Fan, 2017) in many articles have documented the process of extracting every word from API and mashup descriptions and converting them into vectorised values for API-mashup matrix preparation. By recommending API services for the newly created mashup, the recommendation system should learn from the interactions between users and APIs or mashups before the recommendation can begin (Yu, Wong & Chi, 2017). To address the issue with search engines, performing the natural language processing (NLP) actions can benefit the mashup developers to find suitable APIs quickly.

Figure 2.4 shows the processes of recommending the suitable APIs by utilising the MF algorithm with regularisation mechanism. In the case of Fletcher (2019), he used the hierarchical Dirichlet process and Jensen-Shannon divergence methods of calculating the similarity values of both web APIs. However, both methods of calculating the web API similarity values are outside the scope of this study. There are many ways to compute similarity values. The method suitable for this study is to utilise the TF-IDF (term frequency - inverse document frequency) method by (Yao et al., 2018). The TF-IDF method is simple to compute the frequency of every word in a document.

In respect to user convenience, the CF method is proven useful for referring the user

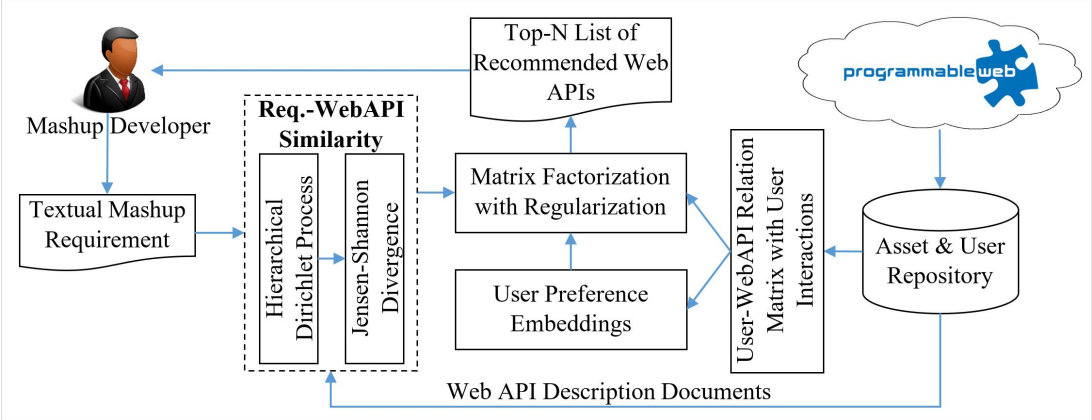

Figure 2.4: Diagram showing the proposed framework recommending suitable APIs (Fletcher, 2019).

preference embeddings, which stores the relationship data was stored. It is called the functional service recommendation which the service requirements should satisfy the needs of users (Hao et al., 2017). When a user enters the desired keywords, the system computes the similarity from the search terms, locates the API by similarity results, utilise MF for learning missing values and finally, the APIs were recommended to the user (H. Wang et al., 2018).

2.4.2 Social Recommendation

When social networks such as Facebook and Twitter emerged since the 2000s, other websites have been implementing the method of allowing the users to create their accounts and review items they have bought on the Internet. In other words, the influence of social media such as Facebook and Twitter shaped the future of implementing the social recommendation systems (Tang, Hu & Liu, 2013). The fast-growing development of social networks contains a variety of information of any users, including the relations between users, posts, comments and tags (W. Zhang, Liu, Xu & Jiang, 2019). Regarding posts, comments and likes, every activity has the date and time recorded every time a user starts publishing an item on social networks (Madani, Erritali, Bengourram &

Sailhan, 2019). Besides, the social network can also add attributes of a user profile such as the location, educational attendance, groups, income, occupation, gender and age (S.-D. Liu & Meng, 2015; Xiang, Neville & Rogati, 2010). These attributes about a user in a social network provide valuable information for the social recommendation, and they improve the accuracy especially when the data is sparse (Ma, Zhou, Liu, Lyu & King, 2011; Shi et al., 2014). Depending on the provider, recommender systems may utilise user profile attributes to enhance recommendation results for a particular user based on its social space.

Regarding social information, the three main goals for researchers to analyse and implement recommender systems are (Bobadilla et al., 2013): (1) Build new recommender systems, (2) exploit the relationships between users and items and (3) provide more accurate predictions from machine learning. As a result, the influence of social networks allows shopping websites to store information about the users and the items they bought or use. This approach is different from the standard recommender systems as the system relies on friends to make a recommendation based on the connection between users (Guo, Wen & Wang, 2018). Both Figure 2.5 and Table 2.1 provides

	V1	**V2**	**V3**	**V4**
U1	5	?	?	?
U2	3	?	1	?
U3	?	?	?	5
U4	?	4	?	?
U5	3	?	2	?
U6	?	3	?	?
U7	?	4	?	1

	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**
U1	0	0	0	0	1	1	0
U2	0	0	0	0	1	0	0
U3	0	0	0	1	1	0	0
U4	0	0	1	0	1	0	1
U5	1	1	1	1	0	0	0
U6	1	0	0	0	0	0	0
U7	0	0	0	1	0	0	0

Table 2.1: Both matrices of the user-item ratings and friendship relations.

a relationship diagram and their matrices to show the relationship between users and items. An excellent example of social recommendation is Guo, Ma, Chen and Jiang (2014) provided an example of the Chinese Weibo's social networking architecture which users can follow the pages denoted as items to receive updates. With the social

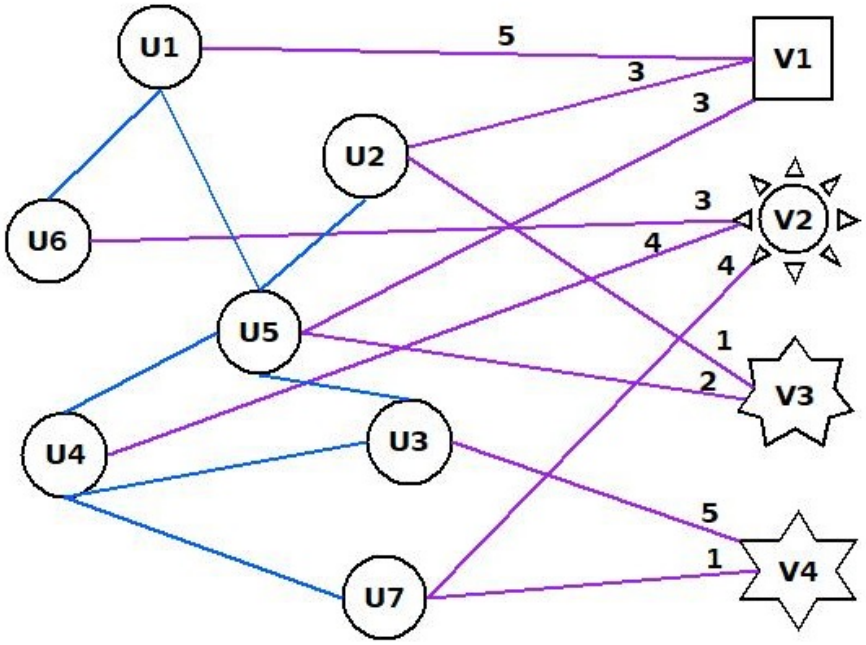

Figure 2.5: The relationship between users and rated items.

recommendation method, it can utilise both social relations and the item ratings to produce implicit conclusions for recommending the items for a friend circle (S.-D. Liu & Meng, 2015). The matrix on the left of Table 2.1 contains the user and item ratings from 1 to 5, and on the right, the friendship values are only 0 (no connection) or 1 (linked).

In social circles, the relationship strength between users is unequal in different friend groups (Yang, Steck & Liu, 2012). For example, User *A* has 5 friends in any circles: *B*, *C*, *D*, *E* and *F*. Users *A*, *B*, *C*, *D* and *E* belongs to Circle *A*, and Users *F*, *G*, *H*, *I* and *J* belongs to Circle *B*. Social networking can have strong and weak ties between users. Also, users with strong ties belong within a social circle, and inter-circle relations are weak (X. Wang, Lu, Ester, Wang & Chen, 2016). A social circle can have categories that allow discussion of a single topic only, e.g. Circle *A* concerns about music and is different from Circle *B*, which discusses computer sciences. In reality, social circles are irrelevant because a user can have multiple categories in its whole

social space (Yang et al., 2014). Thus, recommender systems are not concerned with recommendations within or outside the social circles.

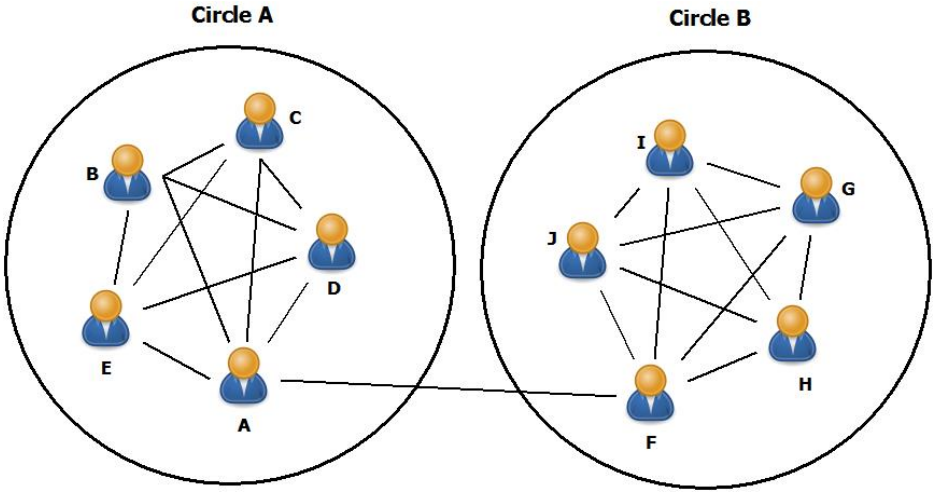

Figure 2.6: Social circles with strong and weak ties.

Social recommendation is thought to be reliable recommender systems. However, the user's perspective towards others can ignore the importance of the recommendation mechanism (Guo et al., 2014). The advantages of social recommendation are (H.-F. Liu, Jing & Yu, 2018):

- **It increases the accuracy for recommendation:**

  Compared to utilising the traditional methods based on the items only, integrating social information can greatly enhance recommendation for users.

- **It can effectively overcome the cold start issues:**

  The traditional recommender systems that store the sparse data can result in uneven distributions. Therefore, social recommendation can partially solve the cold start problem when new users are added.

- **It increases the trustfulness of the recommendation results:**

  With recommendation based the social or trust information, the system is powerfully persuasive that a user can trust the outcome of the recommendation.

- **The ease of understanding of the recommendation results:**
  Relying on the social information for the recommendation on the item can significantly enhance the user-friendliness of recommender systems.

In regards to the advantages of utilising social recommendation, it is a useful function to recommend items for users.

The cold start issue in recommender systems is the major problem for the recommendation algorithms not being able to learn the data for consistent results implicitly. Since new users or items are continually increasing, the system may only predict results from the existing data. Therefore, any new users without collaborative information like no prior purchases, ratings or interactions can produce inconsistent results following prediction (Gantner, Drumond, Freudenthaler, Rendle & Schmidt-Thieme, 2010; Madani et al., 2019; Verma et al., 2013). The pure CF algorithm may seem to alleviate the problem, but, it requires direct input from the existing users (Sedhain, Menon, Sanner, Xie & Braziunas, 2017). By asking the other users for their help, it may not worth the effort when new users are constantly increasing. By overcoming the cold start issue, Sedhain, Sanner, Braziunas, Xie and Christensen (2014) proposed an idea of integrating the user and item matrices in the equation that have three variables: U is the user matrix, I is the item matrix, and P is the user's personal information. The user's data could contain demographic traits such as gender, location, age, friends, likes and comments. Thus, with the usage of the user's metadata, it could be similar to the content-based filtering method.

2.5 Conclusion

This chapter provided a literature review of the recommendation methods and algorithms. We have covered the filtering methods that affect the quality of the filtered results in

different filtering algorithms. The filtering algorithms have provided an essential understanding of how the recommendation system works differently. The similarity equations in CF has the potential for manipulating the API description list to output the values. The content-based filtering is not part of our study, but it does increase the knowledge base for future research in conjunction with CF. The recommendation methods were introduced with service recommendation and social recommendation. Service recommendation has impacted the potential of this study because the information can provide some hints to understand how the framework recommends suitable APIs. Although social recommendation is unrelated to this study, it allowed us to explore the wide variety of methods that are used for recommendation. Finally, we have explained the MF methods and their variants in this chapter.

Chapter 3

Research Method

3.1 Introduction

In this chapter, we will begin with an overview of the research process. We will discuss how the dataset is preprocessed for the experiment, the process of extracting the words from the API (application programming interface) description list and the calculation of the similarity values. In addition to preprocessing, we will discuss the natural language processing (NLP) libraries in this chapter. We will also provide the equations that will show how the experiment should proceed and the evaluation metrics to test the data reliability for this research.

3.2 Approach Overview

As the purpose of this study is to produce a prediction result for API mashup relations, the methodology for this research is quantitative because it is intended for the project that will produce results that are utilised by numerical methods (Ahmad, Wasim, Gogoi, Srivastava & Farheen, 2019).

The matrix factorisation-based (MF) service recommendation process is based on

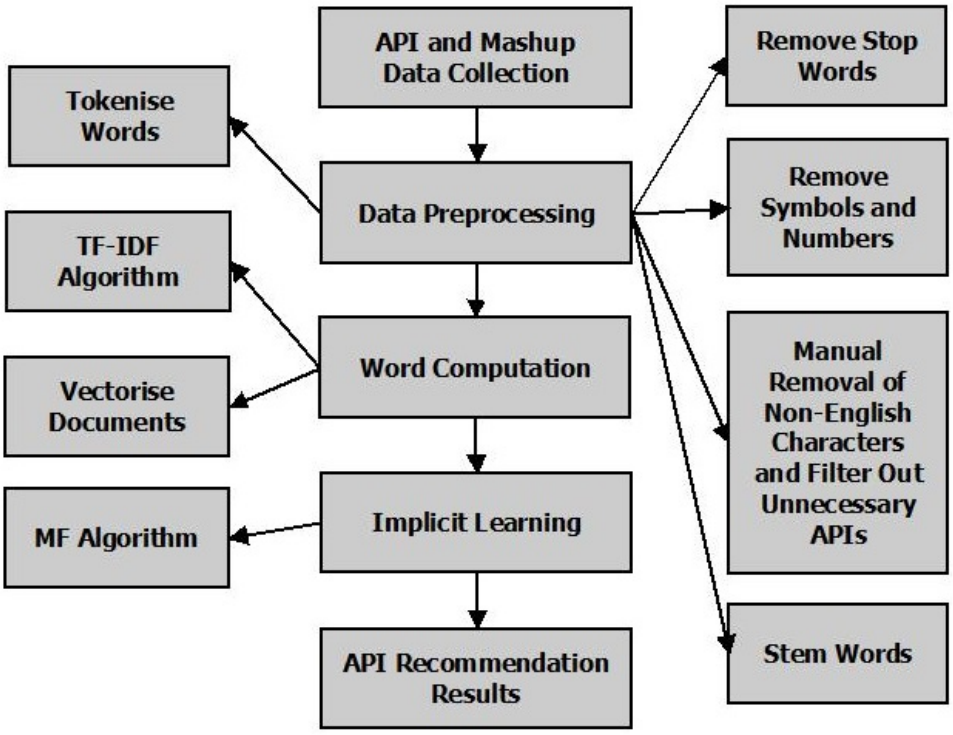

Figure 3.1: Flowchart of overall processes.

the paper from Yao et al. (2018) titled *"Mashup Recommendation by Regularizing Matrix Factorization with API Co-Invocations"*. The flowchart in Figure 3.1 depicts the overall process of converting words into recommendation results. The five main steps of the approach overview are described:

1. **API and Mashup Data Collection:**

   The data collection process will be described in Section 3.3.

2. **Data Preprocessing:**

   It is the most crucial aspect of handling the data source collected from ProgrammableWeb. It consists of word manipulation processes not limited to tokenising and stemming words, filtering out unnecessary APIs, omitting and substituting characters with diacritics, removing stop words, symbols and numbers. However, the drawback is the preprocessing procedures are vulnerable to accidental

deletions, so care must be taken to avoid it.

3. **Word Computation:**

   The algorithm used to calculate the frequency rate of every word is term frequency - inverse document frequency (TF-IDF). The frequency of every word can measure the rate of occurrence in a document. It will be used to determine the level of similarity between APIs and mashups. Since the conversion from corpus to numbers are essential for producing the recommendation results, we have to save the data in the CSV (comma-separated values) format.

4. **Implicit Learning:**

   We use the machine learning method of stochastic gradient descent (SGD) rules to learn the data from the explicit invocations. Before the SGD learning method is used, we calculate the latent variable by the generic coordinate ascent method by updating from the required parameters. Afterwards, implicit learning fills up the gap between the relationship between APIs and mashups.

5. **API Recommendation Results:**

   The results will be presented as an $m \times n$ matrix following the implicit learning procedures. However, this procedure is not the end as the results are not likely to be reliable for the first time. The repetition of learning processes is necessary for improving the results and is not limited to trial and error, such as the learning rate and lambda variables.

3.2.1 Data Preprocessing

To proceed with the data preprocessing, we use the API dataset and the API-mashup invocation list to implement the functions needed to achieve the results from corpus

mining. One of the most important libraries for this process is the NLTK[1] libraries provide many functions for NLP. Additionally, many NLTK libraries support the processing of every word in a document such as Porter Stemmer, Punkt Tokenizer, Web Text Corpus, WordNet, Stopwords Corpus and many more. Therefore, specific word processing tools will be useful for this project. The whole process of extracting titles and descriptions from the API lists dataset and converting them into vectorised form is explained (Hussain et al., 2018).

1. **Manual Removal of Non-English Characters and Filter out Unnecessary APIs:**

   The removal of non-English characters is a trial-and-error process. Since the non-English Latin character removal process requires us to search for the culprit line number, the program should throw an exception when stemming words. As the specific line numbers were identified, we edit the API descriptions file and find any words that contain any special or non-English characters. For omitting the unlisted APIs, there are no libraries available for filtering out unnecessary APIs based on the mashup-API invocation list. The only way is to write a code that will loop each line of API names and output the APIs and its descriptions into the list. This task is tedious because the results may produce repetitive APIs that will cause us to repeat the process further. Therefore, further processing is necessary to prevent repetitive names on the list.

2. **Remove Stop Words:**

   Removing stop words such as 'I', 'is', 'are', 'am', 'when', 'he', 'she', and 'without' is a necessary process to ensure no meaningless words are counted in the experiment. The Stopwords Corpus library is part of NLTK that is available on the Python[2] programming language.

[1]NLTK website: http://www.nltk.org/
[2]Python website: https://www.python.org/

3. **Remove Symbols and Numbers:**

   Numbers and symbols such as '&', '@', '+' and '$' do not represent any linguistic meaning in NLP. Consequently, they are excluded from the MF process.

4. **Stem words:**

   Any words formed in one of the linguistic processes has appended with the suffixes as part of the natural language procedures (Singh & Gupta, 2019). It is a crucial process to trim the words to its base form, for example, the word *legalize* becomes *legal*, *studies* becomes *studi*, *apples* becomes *apple* and *sensational* becomes *sensat* (Porter, 1980). However, the drawback of the Porter Stemmer library is it does not recognise the non-American English -ise suffix (-ize used in American English) on the word stemming list. Despite the drawback, the Porter Stemmer library is thought to do an adequate job for trimming word suffixes and plurals into the lower form in the English language.

3.2.2 Word Computation

Converting corpora into numbers is part of the NLP method that calculate the similarities between connecting words. Two processes need to be done before the experiment can begin.

1. **Calculate TF-IDF:**

   The procedure shows that all words in a document can be converted into numbers signifying the frequency of each word appears in an API list and its descriptions. In other words, it calculates the TF-IDF values on all words in each list of API name and description. Each word is paired with the TF-IDF values within an API list. Therefore, it can be converted into a list of vectors.

2. **Vectorise documents:**

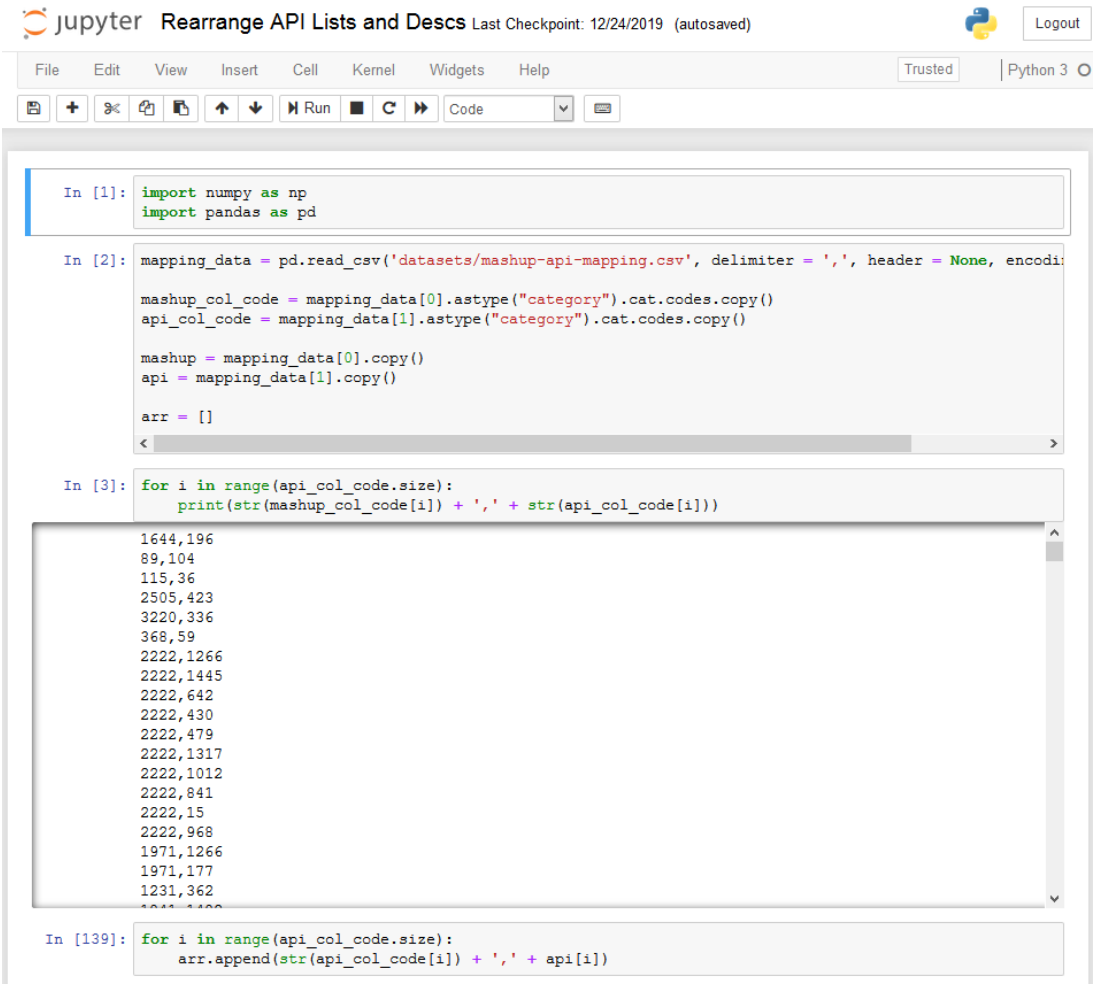

Figure 3.2: Illustration showing the procedure of manual data preprocessing for manually omitting unnecessary APIs. Screenshot by author.

The list of numbers contained in the list of documents will become vectorised to ensure that every tokenised word has been assigned as a one-dimensional vector. This process is part of NLP because it converts words into vectors. Many libraries will convert words into vectors such as Word2Vec, Dep2Vec, FastText and Glove (Arroyo-Fernández, Méndez-Cruz, Sierra, Torres-Moreno & Sidorov, 2019). All word vectorisation programs mentioned can be used for convenience. However, the Python program allows linking two indices together without the need to utilise the mentioned libraries.

3.3 Data Source

The data source is from ProgrammableWeb. The dataset used for this project contains about 1549 APIs and 6202 mashups. One can easily search the API and mashups information, including those that are no longer supported that requires selecting the tick box. The website also allows developers to upload the APIs, SDKs (software development kit), sample source codes or mashups to the server, and the general users can track the APIs and mashups from the website. We have the dataset containing the list of APIs and its descriptions and the API-mashup invocation list. The screenshot

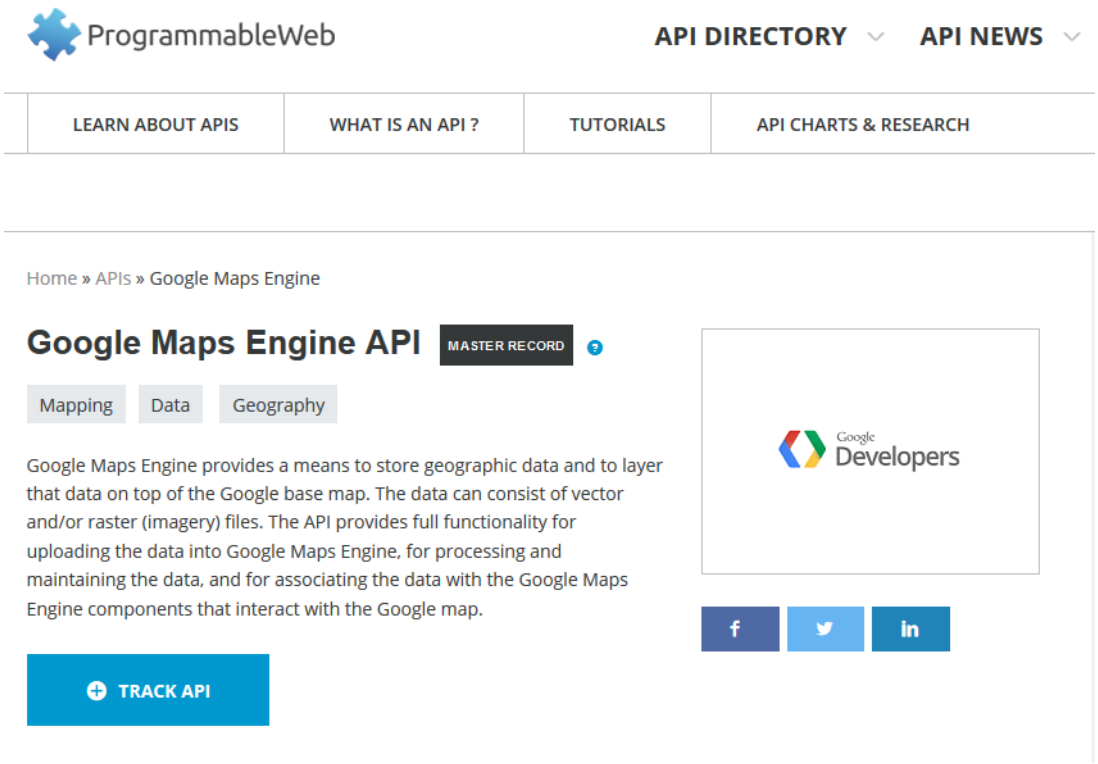

Figure 3.3: ProgrammableWeb. (n.d.). *Google Maps Engine API is stored in the ProgrammableWeb API directory* [Screenshot].
Retrieved from https://www.programmableweb.com/api/google-maps-engine

in Figure 3.3 shows the information of the Google Maps Engine API and described it. On the ProgrammableWeb API profile, the tags were included for categorisation and allowed anyone to share the API information to the social networks.

3.4 Technical Details of the Proposed Method

In this experiment, we will denote the users as APIs and mashups as items. To conduct this, we will use machine learning to apply the techniques based on a journal paper. Yao et al. (2018) provided the equations to build the prototype and conduct the experiment for learning the effectiveness of recommendation for the mashups based on the TF-IDF values of the API-mashup list.

3.4.1 TF-IDF Calculation

TF-IDF calculates the occurrence of a word in a document. TF (term frequency) is the prevalence of a term in a document. The equation of TF is

$$TF(t) = \frac{\textit{Number of t in a document}}{\textit{A combined API title and its description}} \tag{3.1}$$

where t denotes a term in a document. The TF equation looks like the probability that a word may appear in a document, and it is simple to understand. Additionally, IDF (inverse document frequency) is calculated by

$$IDF(t) = log(\frac{N}{DF}) \tag{3.2}$$

where N is the number of corpora, and DF is the frequency of a term. If the DF value is zero, it will cause a divide by zero exception, and the program will stop working immediately. To overcome the problem, the equation should be

$$IDF(t) = log(\frac{N}{DF+1}) \tag{3.3}$$

where it should compensate against returning invalid values that will cause the program to throw an exception. By extracting the terms from the API titles and its descriptions,

there will be a large number of stop words that will interfere with the experiment and the recommendation strength. As previously mentioned in Section 3.2.1, those processes are important to prune unnecessary words and stem them to provide more accurate results.

3.4.2 Cosine Similarity Calculation

The cosine similarity is used for computing the probability of the API similarities based on the TF-IDF values calculated. It also used to dictate the similarity between web APIs and mashup specification (Jain et al., 2015). An API vector is defined as

$$a_i = TF \times IDF \tag{3.4}$$

As a_{ij} represents an API vector, the cosine similarity will be calculated as

$$s_{ij} = \frac{a_i \cdot a_j}{||a_i||||a_j||} \tag{3.5}$$

where $i = \{1, 2, ..., n\}$ and $j = \{1, 2, ..., n\}$ are matrix indices as both sets have the same maximum values. In other words, s_{ij} is a square matrix. This equation is also mentioned in Equation 2.2 in the previous chapter. However, we have to repeat this equation to use a different expression for this experiment.

3.4.3 Model Learning and Increasing the Logarithm Function

The model learning would be utilised by machine learning. The goal for the function is to maximise the logarithm function by updating the variable z_{ij}

$$z_{ij}^{new} := z_{ij}^{old} - \alpha_z \frac{\partial L}{\partial z_{ij}} / \frac{\partial^2 L}{\partial (z_{ij})^2} \tag{3.6}$$

where both first and second derivatives for z_{ij} are

$$\frac{\partial L}{\partial z_{ij}} = \frac{1}{\sigma^2} \times (w_{ij} \times s_{ij} - z_{ij}) + \frac{y_{ij}}{z_{ij}} \tag{3.7}$$

$$\frac{\partial^2 L}{\partial (z_{ij})^2} = \frac{-1}{\sigma^2} - \frac{y_{ij}}{z_{ij}^2} \tag{3.8}$$

Also variable θ_{ij}

$$\theta_{ij}^{new} := \theta_{ij}^{old} - \alpha_\theta \frac{\partial L}{\partial \theta_{ij}} / \frac{\partial^2 L}{\partial (\theta_{ij})^2} \tag{3.9}$$

where both derivatives follow

$$\frac{\partial L}{\partial \theta_{ij}} = \frac{1}{\sigma^2} \sum_{ij \epsilon D} (\frac{1}{\theta_{ij}} \times y_{ij} - z_{ij}) - \lambda_\theta \theta_{ij} \tag{3.10}$$

$$\frac{\partial^2 L}{\partial \theta_{ij}^2} = \frac{1}{\sigma^2} \sum_{ij \epsilon D} (\frac{-y_{ij}}{\theta_{ij}^2}) - \lambda_\theta \tag{3.11}$$

To understand what the variables are from the above equations, z_{ij} is the latent variable, w_{ij} is the estimated weight vector variable, and s_{ij} is the cosine similarity variable. z_{ij} will be initialised from the probability values distributed by $N(w_{ij}s_{ij}, \sigma^2)$. Afterwards, the latent variable of z_{ij} will be implicitly learned as well as θ_{ij}. The learning parameters α and λ controls the pace of the machine learning.

To increase the logarithm function, both z and θ variables determines the value of the logarithm function.

$$max(Z, \theta) = \frac{-1}{2\sigma^2} \sum_{ij \epsilon D} ((z_{ij} - w_{ij}s_{ij})^2 + y_{ij} log(\theta_{ij} z_{ij}) - \theta_{ij} z_{ij}) - \frac{\lambda}{2} W^T \cdot W - \frac{\lambda}{2} \theta^T \cdot \theta \tag{3.12}$$

Variables z_{ij}, w_{ij} and θ_{ij} are square matrices. Note the dot product between W^T and W; θ^T and θ in the equation. Both variables are reshaped as an $m \times 1$ matrix for representing a one-dimensional vector. If both variables are not reshaped to compute the logarithm function, then it is impossible to dot product θ and W vectors into a single number.

While the logarithm equation increases, the W vector can be computed as

$$W^{new} = (\lambda_W I + S^T S)^{-1} S^T Z \tag{3.13}$$

W^{new} is an $m \times 1$ reshaped matrix following the calculation from the existing S and Z variables. The equation will be updated each time after the logarithm function has completed each iteration. The diagram from Figure 3.4 shows the direction of both

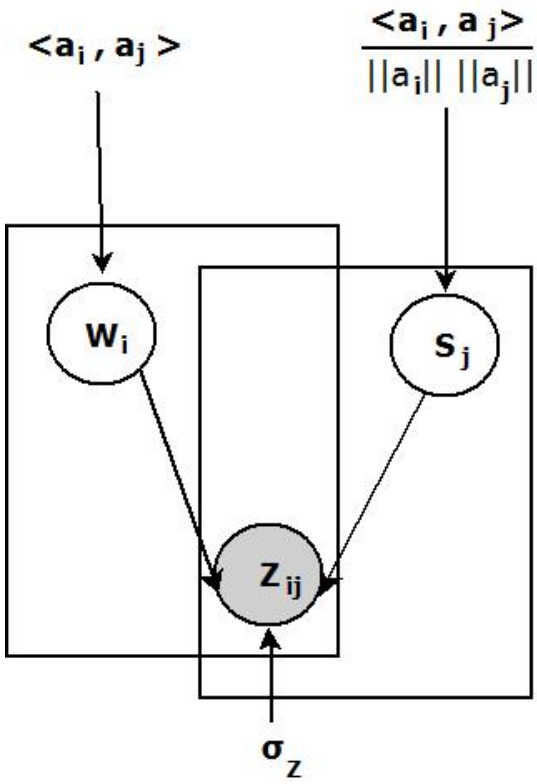

Figure 3.4: Diagram showing the Z_{ij} latent variable learning from S_{ij} and W_{ij}.

variables W_i and S_j have their direct relationship towards Z_{ij}.

3.4.4 Loss Minimisation

Loss minimisation is used for learning the effectiveness of the API recommendation for mashups. The cost function is

$$min(A, M) = \frac{1}{2}\sum_{i=1}^{M}\sum_{j=1}^{N}(I_{ij}(r_{ij} - a_i^T m_j)^2) + \frac{\alpha}{2}\sum_{i=1}^{M}\sum_{b=1}^{P}(z_{ib}||a_i - a_b||^2) + \frac{\lambda}{2}||A||^2 + \frac{\lambda}{2}||M||^2 \tag{3.14}$$

Note the maximum values M and N can be equal or unequal to each other, as denoted that i and j can be different. When the loss function decreases gradually, the lesser the value, the better it is. The variables A and M are optimised by the SGD updating rules to reach their local minima (Koren et al., 2009)

$$a_i^{new} := a_i^{old} + \eta_A(\delta_{ij} m_j - \alpha \sum_{b \in J}(z_{ib}(a_i - a_b)) \tag{3.15}$$

$$m_j^{new} := m_j + \eta_M(\delta_{ij} a_i - \lambda m_j) \tag{3.16}$$

where η_A and η_M denote both learning rates in A and M, $\delta_{ij} = 0$ if $i \neq j$, and $\delta_{ij} = 1$ otherwise. As both variables are used for learning the model of the equation, they are dictating the probability that an API invokes the mashups in an $m \times n$ matrix. The m side of the matrix represents the API rows, and the n side denotes the mashup columns. Figure 3.5 illustrates how the R_{ij} variable is learning from both API and mashup dimensional matrices. However, the R_{ij} matrix does not have any values initiated because the equation from 3.14 does not express the R_{ij}^{prob} variable and we should denote the variable as $r_{ij}^{prob} = a_i^T m_j$.

3.5 Conclusion

This chapter provided methods and specifications of this research. The approach overview provided the overall research processes that allowed us to achieve the API

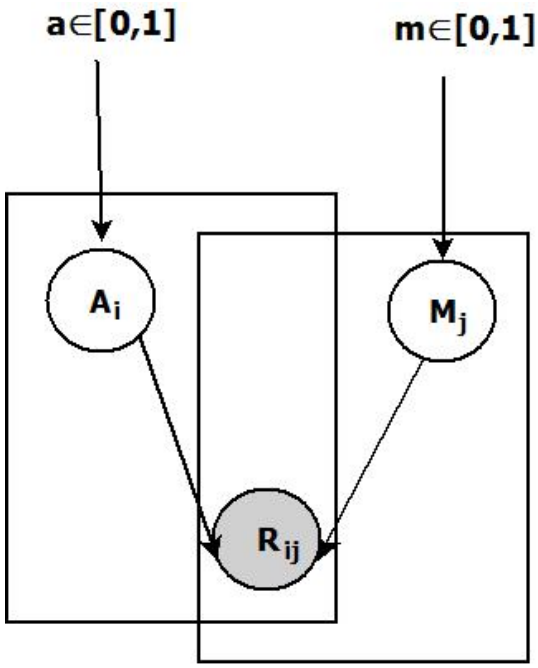

Figure 3.5: Diagram showing R_{ij} is learning from A_i and M_j.

recommendation goals. Also, the data preprocessing procedures were described in detail for the optimisation of the data input. The technical specifications were comprised of equations and evaluation metrics to familiarise the background in the MF method. Additionally, the equations provided in the technical specifications were sourced from Yao et al. (2018).

Chapter 4

Results and Discussion

4.1 Introduction

In this chapter, we will introduce the results from the experiment we had done in this study. We provide the setup specification of the experiment and discuss the details of the procedure. Also, the graphs and tables related to the investigation are provided to investigate the aftermath of the implementation.

We report all results and describe the features and essential aspects of the experiment. Most results in this chapter are in graph formats, and the data are based on the equations in the last chapter we have mentioned in this thesis. First, we introduce the table concerning the top 40 words ranked from the most common to the least. Afterwards, we provide the results about an example of a matrix calculated from the matrix factorisation (MF) algorithm. Additionally, this chapter is also suitable to contribute to the discussion that relates to the experiment.

4.2 Experiments

4.2.1 Experiment Setup

To set up the experiment, we installed the required software on a computer to proceed with the MF experimentation. The computer specifications used for the project were:

- Microsoft Windows 8.1 operating system
- Intel Core i7 6700 CPU
- 16GB of RAM
- 2TB HDD
- NVIDIA GTX 1060 graphics card with 3GB of GDDR5 RAM

As the experiment requires machine learning to implicitly learn from existing API-mashup (application programming interface) relationships, the software installed for this project were Anaconda[1] in Figure 4.1 and Jupyter Notebook by Jupyter[2] in Figure 4.2. The Python programming language was included with the installation of Jupyter Notebook. Python was the preferred language to implement this experiment because not only it became a popular programming language; many libraries are suitable for this project. The libraries in Python such as NumPy[3], Pandas[4] and NLTK does support our work for the MF prototype.

4.2.2 Acceleration of the Performance

Due to a large number of APIs and mashups, the performance is likely to deteriorate. When the system suffers slow performance, it can cause one part of an algorithm to

[1] Anaconda website: https://www.anaconda.com/
[2] Project Jupyter website: https://jupyter.org/
[3] Numpy website: https://numpy.org/
[4] Pandas website: https://pandas.pydata.org/

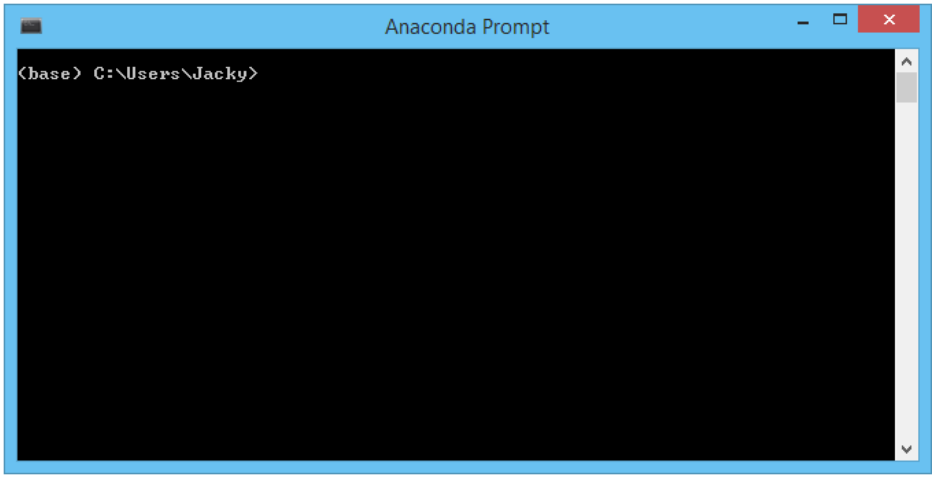

Figure 4.1: Illustration showing the Anaconda prompt window. Screenshot by author.

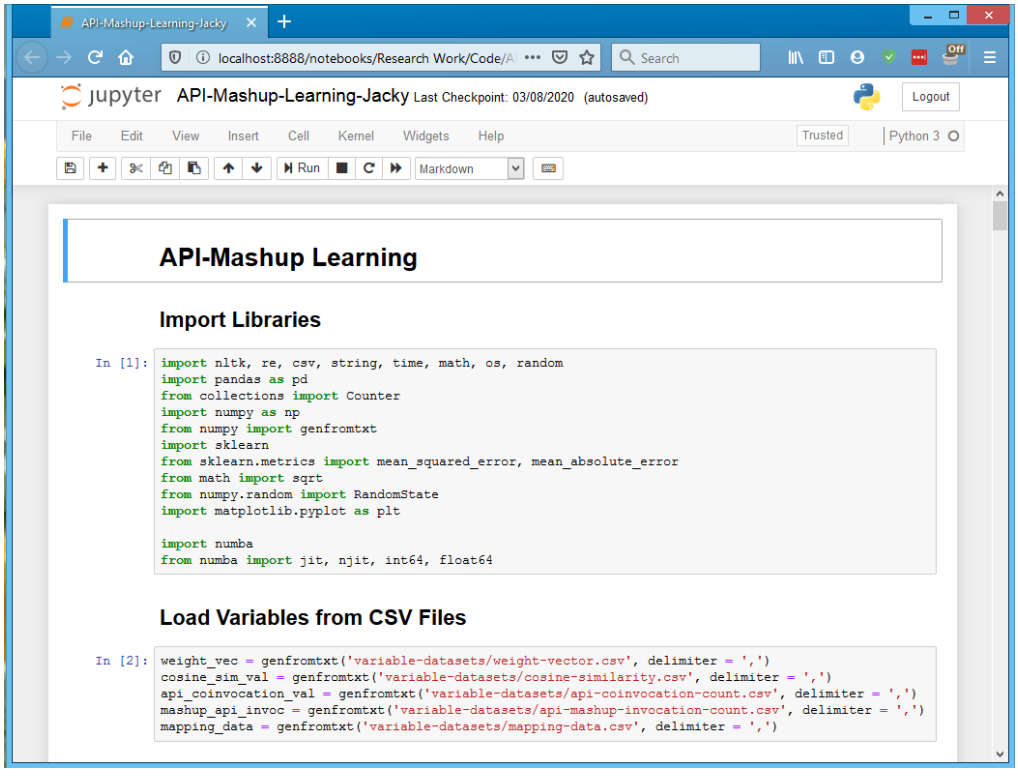

Figure 4.2: Illustration showing the Jupyter Notebook user interface. Screenshot by author.

spend a long time to finish a portion of the process. To alleviate this problem for training with MF, the Anaconda Package called Numba[5] should be installed on the computer. Depending on the specifications of the computer, if the motherboard contains a multicore processor, Numba will utilise them to speed up the process.

The issue with hardware acceleration is the CPU (central processing unit) usage will rise to nearly 100%, and it can stress the processor to work much harder. Also, the system would crash if the RAM (random-access memory) runs out of available space, it will cause the computer to utilise swap memory in the hard disk or solid-state disk. Using a virtual machine or a home computer is recommended for this experiment. Therefore, caution should be taken to prevent crashing the system, especially if one uses Numba on a multi-user server.

4.2.3 Exporting Variables to CSV Files

Saving each data to comma-separated values (CSV) files should be done after preprocessing to prevent the loss of a variable data if the system crashes or there is a power failure. Besides, the slow performance of the data processing can be frustrating to wait until it completes. To ensure the variable data is preserved during the experiment, one can retrieve the saved data if the program starts to throw an exception when an error occurs. Thus, this action should be considered if the research project requires the manipulation of huge data.

4.2.4 Evaluation Metrics

The mean absolute error (MAE) and root mean square error (RMSE) evaluation metrics are used for evaluating whether the training data is underfitting or overfitting. Also, it evaluates how recommender systems can perform its task of producing accurate

[5]Numba website: http://numba.pydata.org/

predictions (Kluver & Konstan, 2014). For both metrics, lower values mean the data has better performance in the experiment.

$$MAE = \frac{\sum_{ij}(r_{ij}^{pred} - r_{ij}^{truth})}{N} \tag{4.1}$$

$$RMSE = \sqrt{\frac{\sum_{ij}(r_{ij}^{pred} - r_{ij}^{truth})}{N}} \tag{4.2}$$

N denotes the number of samples used in both metrics. Despite the better performance in lower values, overfitting the data can produce very accurate results in prediction. The last thing we want is to get both MAE and RMSE values down to nearly zero. The near-zero values are not what we want for the experiment. On the other hand, underfitting the data can also produce inaccurate results.

4.2.5 How the Experiment was Conducted

The experiment was conducted solely on a computer with the required applications installed. To record the results, we have utilised the Matplotlib[6] library to plot the graphs from the data. As previously mentioned in Subsection 4.2.3, exporting the results is essential to prepare for application freezes resulting in data losses. When conducting the experiment, we used the trial-and-error method to set up the variables needed to fine-tune the rate of learning. To ensure the learning processes are smooth, we have to let the learning progresses to print out into the screen. Afterwards, the graph is created by the Matplotlib library.

The objective for recommending unrelated APIs for a mashup application is to solve latent variables required for learning the relationship between the two. Furthermore, we will use the Coordinate Descent method, which is similar to stochastic gradient descent (SGD), to minimise or maximise latent variables. The illustration of the SGD graph

[6]Matplotlib website: https://matplotlib.org/

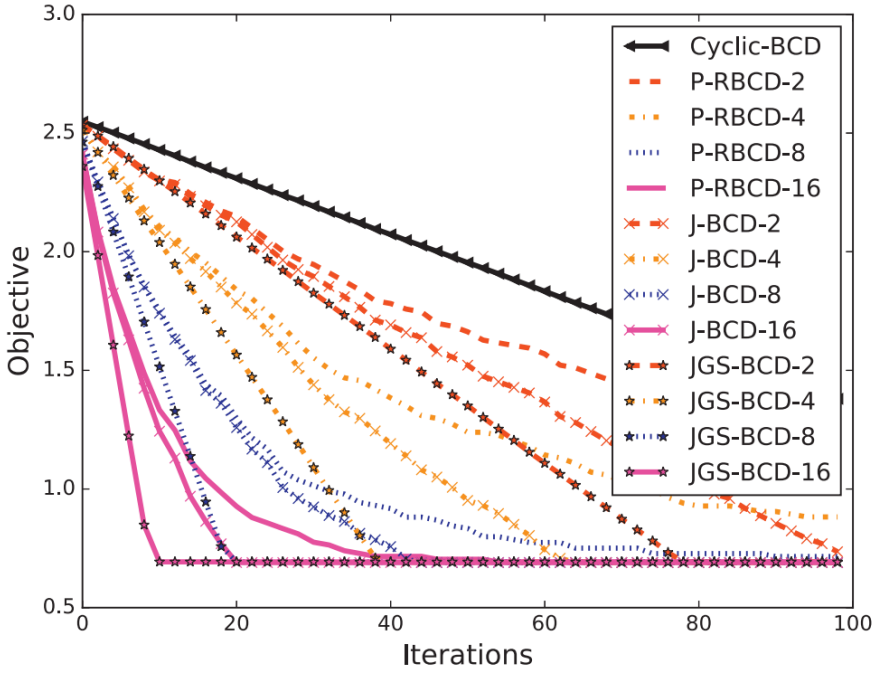

Figure 4.3: The example graph depicting the coordinate descent method (X. Wang et al., 2018).

provides an example of how the machine learning method looks like when attempting to find the minimum or maximum of the function in Figure 4.3.

The θ and Z latent parameters should be minimised to maximise the logarithm value of the function, according to Yao et al. (2018). Also, the A and M variables should be learned following the application of latent parameters. We have attempted to try out different learning rates to observe the quality of the latent parameters. With the process of trial-and-error, we can observe the details of minimising the latent parameters by outputting the values of θ, Z, A and M variables. Outputting the values of all parameters are intended for testing purposes only and will not be recorded on the results. While we were choosing a suitable learning rate, we have encountered many values bouncing upwards or downwards quickly. Finally, when the convergence point is stable, we can use them as our results.

4.3 Word Extraction Results

When we extracted the words from the dataset, we processed every word during the preprocessing phase. The TF-IDF (term frequency-inverse document frequency) is most commonly used for converting words in each document descending from the most frequent words to the least. We have used the TF-IDF method to vectorise words from the API and the description dataset.

As the program began to process every word, including title, description and tags in the API list, the important process is to remove any stop words. After we have removed the stop words from the API description list, the meaningless words will not interfere with the MF experiment. All documents in the dataset contained punctuations, so they were removed to prevent meaningless recommendations.

Word	Document Frequency	Word	Document Frequency
api	1108	avail	248
u	637	xml	239
service	586	restful	232
user	554	respons	219
data	505	manag	212
allow	494	base	200
provid	493	platform	195
access	480	also	195
applic	470	function	195
developer	463	social	193
web	366	new	191
includ	362	onlin	188
inform	359	website	187
site	338	call	183
search	300	content	183
format	283	tool	180
integrat	275	custom	179
json	273	list	177
description	273	return	175
create	268	time	172

Table 4.1: Top 40 words according to the document frequency (DF) values.

Note that every row in Table 4.1 does not have any stop words after preprocessing. The most frequent word from the dataset is 'api' because the word is frequently mentioned in most descriptions. However, the issue with the word 'api' is that it will match different types of API in various categories, and it could ruin the recommendation results. Most words in the top 40 list are related to the Web and the computer science vocabulary which can hinder the recommendation of the APIs for mashups. One of the root issue with the API descriptions listed in the ProgrammableWeb directory is some of the APIs are outdated or discontinued, which can hinder the actual development of the mashups. In addition, mashup developers can spend more time attempting to find suitable APIs that are not abandoned. Another issue is once an API is discontinued, the message associated with the description like *"this API appears to be no longer available"* can affect the recommendation results. Therefore, the discontinued APIs are not going to provide consistent results if we use it in an actual experiment.

4.4 User-Item Matrix Results

Let R be the user-item $m \times n$ matrix where the row of users represents APIs, and the column of items are mashups. Each matrix element contains a binary value, which 0 means the API does not invoke the mashup and 1 otherwise. The API-mashup matrix specifies the relationship between each API and mashup and transposing it does not affect the contents of it. However, the matrix is more sparse, and as a result, there are many zero values than ones.

When MF is introduced, machine learning can implicitly apply the likelihood that an API is invoking with a mashup. If that is the case, we introduced another $m \times n$ matrix as R^{prob}, which we have once mentioned in Section 3.4.4 in the last chapter. We selected the first 8 APIs and 5 mashups to demonstrate the values in the matrix R^{prob}.

All values in the matrix R^{prob} as shown in Table 4.2 were implicitly learned from MF.

	#API Christm	#Ask4Stuff: WorldCat Twitter Searc	#BeerMap - The Top 2,500 Beers on Twitter	#LinS Stats on Your LinkedIn Social Gr	#S42AT
#blue	0.2665	0.3006	0.2529	0.2847	0.2206
.tel	0.208	0.2606	0.233	0.2275	0.2484
123 ShopPro	0.2095	0.2541	0.1861	0.2397	0.2036
123Contact Form	0.2384	0.3045	0.2294	0.2563	0.2643
12seconds.tv	0.244	0.3078	0.2647	0.2929	0.2489
140 Proof	0.2654	0.3377	0.2778	0.2792	0.2801
18amail	0.2202	0.2667	0.2219	0.2223	0.2377
1Map	0.2278	0.2699	0.2134	0.2219	0.2256

Table 4.2: The R^{prob} matrix after MF learning.

Additionally, not all elements in the matrix can reach above 0.5 because the data were sparse. While the vast majority of the elements contains the values from approximately 0.09 to 0.3, the most common words from Table 4.1 have affected the implicit learning of the R^{prob} matrix. Notice that the words 'api', 'u' and 'service' are the main influences coming from the descriptions of the API list. Therefore, they can increase each API's relationship with every mashup.

Regarding the sparsity of the data, we found that it is the most common problem for recommender systems to recommend APIs for mashups. By looking at the first section of the R^{prob} matrix, we noticed that every value is nearly in range with the values from 0.2 to 0.3.

4.5 Experiment Results

4.5.1 Logarithm Function Results

In machine learning, we used the SGD method to optimise the model and maximise the logarithm value in each iteration (Koren et al., 2009; Yao et al., 2018). The Newton method in machine learning continued to find the local minimum until the next value increases, or it has reached convergence.

The parameters for maximising the logarithm function were $\lambda = 0.5$, $\alpha_Z = 0.005$, $\alpha_\theta = 0.05$, and $\sigma = 1.7$. Also, the maximum number of iterations was set to 400. The α parameters signified the learning rate of the SGD method in both Z and θ. α_Z and α_θ had been set a different amount of learning rates, and the reason is θ was slow to reach its convergence point. If the α_θ learning rate is set too low, θ may not reach its convergence before the logarithm function reaches the maximum.

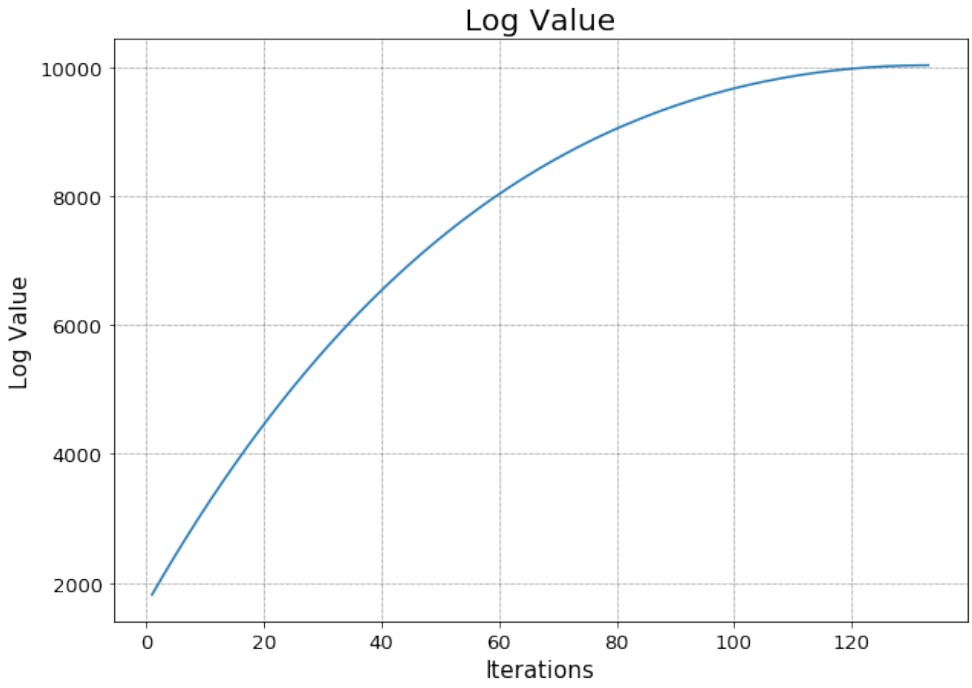

Figure 4.4: Logarithm value over the number of iterations.

According to Figure 4.4, it did not resemble the actual logarithm graph. The

Logarithm function maximisation has reached 134 iterations despite setting the limit to 400 iterations and the value of approximately 10,027. The line has curved slightly steeply upwards as it is a good sign the logarithm value increases rapidly until around the 80th iteration where it started to slow down. In the 80th iteration, it was a sign that shows the learning of the Z variable has begun to decelerate. Unlike Z, the θ variable may not have the potential to reach the convergence point even the application of a large number of the learning rate. As it reached nearly 115th iteration, the logarithm value slowly increased until it reached convergence. From the starting iteration, the initial value of the Log value was almost in 2000.

4.5.2 Loss Function Results

The loss function regards the implicit connection of the other APIs as it should increase the probability that the API may invoke other APIs as close as possible. The value of the loss value should be minimal after training the A and M variables to find the local maximum. In this case, the 90% training and 10% testing ratio will be used on the result.

The parameters we used in the cost function were $\lambda_A = \lambda_M = 0.5$, $\eta_A = \eta_M = 0.00005$, $\alpha = 0.0001$ and $\lambda = 1$. We also set the maximum number of iterations as 30. To justify the smaller values in η and α values, they were assigned smaller amounts because the learning in both A and M variables are very rapid, and it can produce inconsistent results. In other words, faster learning of both variables can cause all values to increase suddenly, thus ruining the potential to reach their minima. The λ and η values applied to both A and M are equal; therefore, it was unnecessary to repeat the same values in different variables. As shown in Figure 4.5, the loss value has decreased a little from approximately 5486 to 5062. As a result, the date is not underfitting nor overfitting.

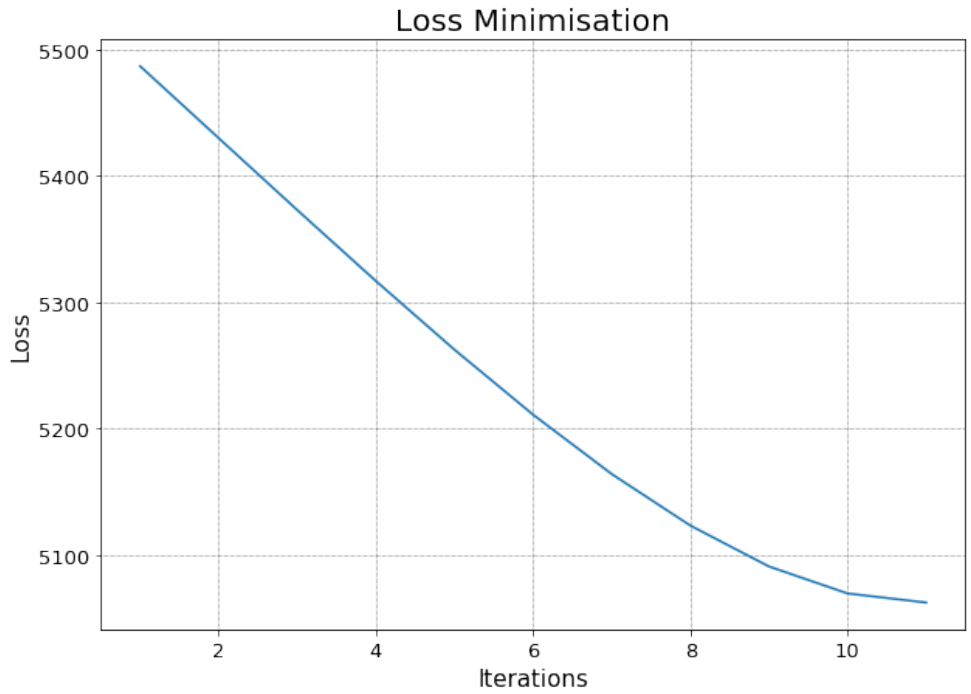

Figure 4.5: The graph depicting the loss value over the number of iterations.

4.6 Evaluation Metrics Results

We compared the evaluation metrics between the different training and testing set ratios. For the training ratio, it began from 90% and decreased it to 50% by the multiple of 10. For the testing ratio, it started at 50% and increased it to 90% in the sample multiple as the last sentence. We utilised the MAE and RMSE values for this section of this chapter.

4.6.1 MAE and RMSE Training Graph

During training to reduce the loss value, the MAE and RMSE were implemented to view the snapshot of detailed training in reducing both values.

Both Figures 4.6 and 4.7 had similar lines in the experiment. Both figures had parallel lines in each graph. The curves in both graphs were decreasing linearly in each iteration, and the maximum iterations in the experiment were 12 iterations.

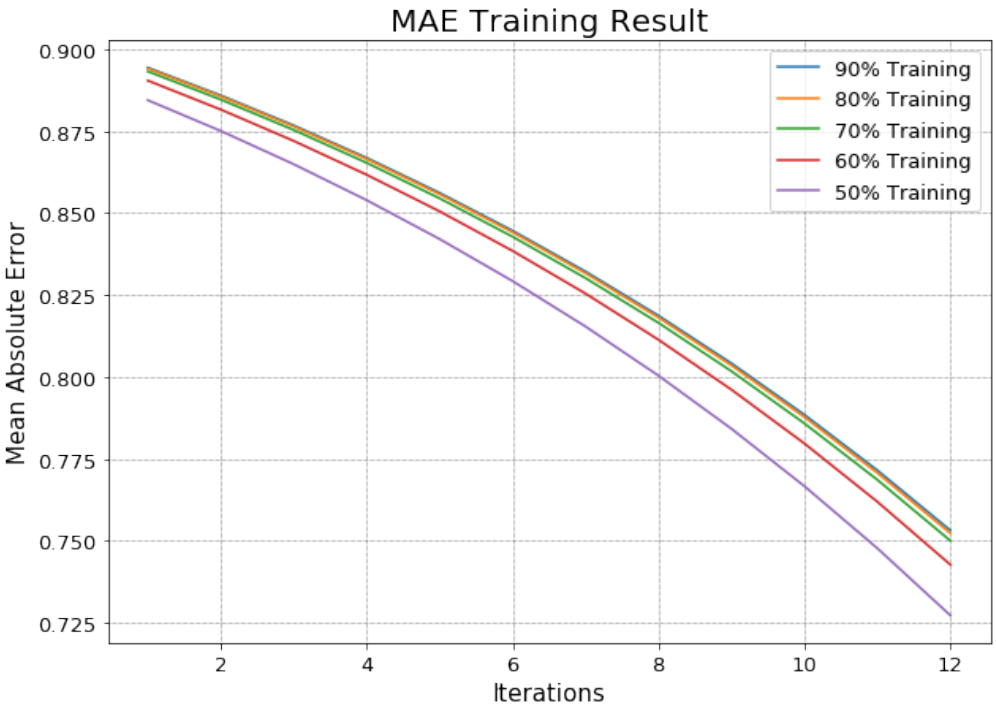

Figure 4.6: MAE training graph.

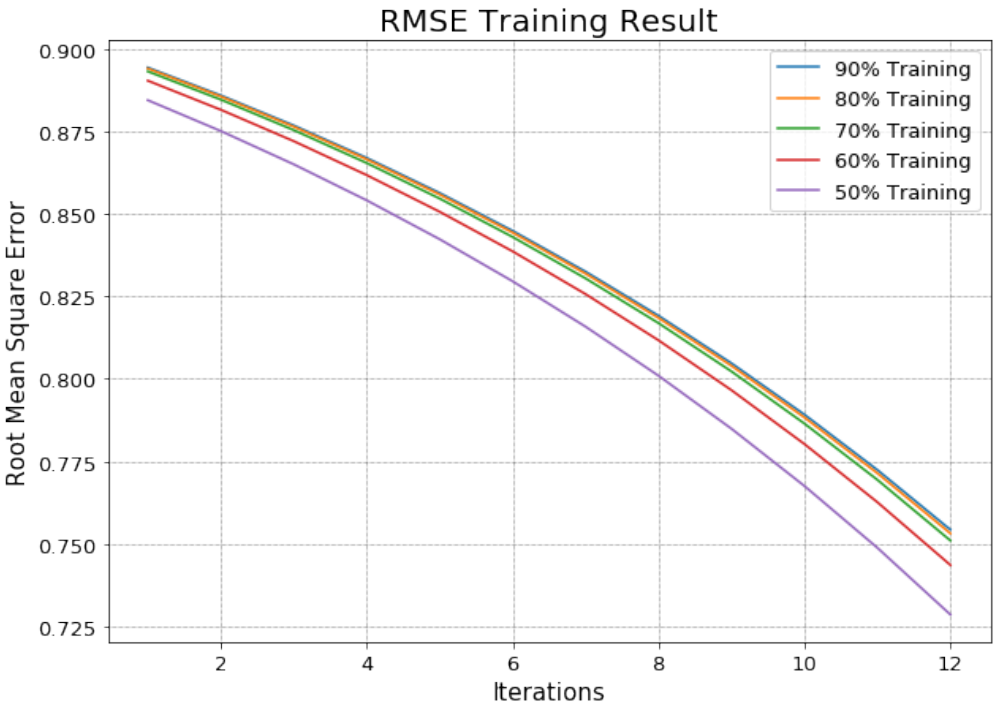

Figure 4.7: RMSE training graph.

4.6.2 MAE and RMSE Training Results

In conjunction with training the loss values in different training ratios, the MAE and RMSE also should be implemented during training in the experiment to examine if the data was overfitting or underfitting.

MAE and RMSE Training Final Value Results					
Training Percentage	90%	80%	70%	60%	50%
MAE	0.7715	0.7707	0.7686	0.7619	0.7478
RMSE	0.7724	0.7714	0.7694	0.7626	0.7488

Table 4.3: MAE and RMSE values in training and testing percentage ratios after training the loss value.

In Figure 4.3, the 50% training ratio had the lowest MAE and RMSE values in the last iteration compared to the highest percentage ratios, which contained large values. If all values had large difference values in all training ratios, then it indicated that there are issues with the experiment. In this case, the difference between all values was relatively small, which means the experiment has done correctly. In terms of data underfit and overfit, the prediction from R^{prob} matrix produced consistent results from the MF training. Although both values should have lowered a little further to produce reliable results, this is a sign of machine learning doing its job to reduce the value of the cost function.

4.6.3 MAE and RMSE Testing Results

In contrast with the training results, we illustrated the testing ratio to verify if the training data was consistent with the testing data.

According to Table 4.4, all the MAE and RMSE values in any training ratios produced the value from 0.7338 to 0.7559. Thus, this proved that the testing data that compares with the training model had worked properly.

Calculation of MAE and RMSE Results Given the Testing Data					
Testing Percentage	10%	20%	30%	40%	50%
MAE	0.7550	0.7517	0.7407	0.7338	0.7489
RMSE	0.7559	0.7525	0.7416	0.7349	0.7498

Table 4.4: MAE and RMSE values evaluated from the testing data.

4.7 Issues Encountered During Prototype Implementation

4.7.1 Equation Flaws from the Journal Article

The paper mentioned in this thesis was Yao et al. (2018) has several errors in the equations used for the experiment. All the derivatives shown on page 7, equations 13, 15 and 16 on the paper has incorrect expressions. The incorrect equation 13 of the first derivative for both variables z_{ij} and θ_{ij} are

$$\frac{\partial L}{\partial z_{ij}} = \frac{1}{\sigma} \sum_{(ij)\epsilon D} ((w_{ij} s_{ij} - z_{ij}) + \frac{y_{ij}}{z_{ij}}) \tag{4.3}$$

$$\frac{\partial L}{\partial \theta} = \sum_{(i,j)\epsilon D} (\frac{1}{\theta} y_{ij} - z_{ij}) - \lambda_\theta \theta \tag{4.4}$$

Part of the second derivative of z_{ij} in equation 15 is

$$\frac{\partial^2 L}{\partial (z_{ij})^2} = \frac{1}{\sigma^2} \sum_{(i,j)\epsilon D} (1 + \frac{y_{ij}}{z_{ij}^2}) \tag{4.5}$$

Part of the second derivative of θ_{ij} equation 16 is

$$\frac{\partial^2 L}{\partial (\theta_{ij})^2} = - \sum_{(i,j)\epsilon D} (y_{ij} \frac{1}{\theta^2}) \tag{4.6}$$

Concerning z_{ij}, Equation 4.3 should not have the summation after the $1/\sigma$ coefficient. The reason is adding all of the elements in the first partial derivative can interfere with

the prediction and the z_{ij} latent variable results. Notice that the $1/\sigma$ coefficient is not squared. Therefore, the summation of the first derivative has caused the program to slow down, although if we installed Numba. Finally, the second derivative of z_{ij} indicated in equation 4.5 is not correct, so we have provided the correct second derivative shown in equation 3.8 in the last chapter.

By making θ_{ij} as the subject, the first derivative is correct except for the θ expression as it should have the ij appended next to it. Equation 4.6 should have $1/\sigma^2$ appended next to the summation coefficient, and the minus sign next to y_{ij} inside the summation expression. Additionally, the $-\lambda_\theta$ term should have been appended next to the equation. Therefore, we have provided the correct functions in equations 3.10 and 3.11 in the last chapter.

The mistakes of the equation on the paper has caused us to investigate and figure out what causes the program to slow down or freeze. We spent much time for the trial-and-error methods to swap variables, substitute different values or introduce the extra variable that will sometimes alleviate the issues. In other times, introducing corrections may produce unreliable results. As the summation of the variables for the Z variable learned using the SGD method caused the program to slow down, we have removed the summation. For both Z and θ variables, the authors should believe that they have forgotten to append the learning rate next to the dividing derivatives. If we did not include the learning rates, the program would assume that the learning rate is 1, and it will experience the values are bouncing back and forth. Finally, we overcame the issue by adding the learning rate for both θ and Z variables and testing the suitable values for the α variable. As a result, the program has started to work flawlessly.

4.7.2 Human Errors Lead to Unintentional Data Overfitting

During training to reduce the cost function, we have encountered situations where both MAE and RMSE values were around zero values, which it signals the data were overfitting because of the programming error. While we understand the lower both MAE and RMSE values are, the better they are in terms of performance. To emphasise the data overfitting issue, we have mistakenly plugged the values of the R matrix (all elements were zeros and ones) and the R^{pred} matrix in the MAE and RMSE equations. As a result, it caused the calculations to produce nearly zero values of MAE and RMSE values, and both results were mistaken as so-called very remarkable values.

We overcame the accidental data overfitting issue is we assigned all ones to the R_{pred} variable of the MAE and RMSE equations. We realised the R_{pred} variable is set to predict that the probability of every API-mashup relationship values should be a little scattered outside the regression value. Once we solved the accidental overfitting problem, both MAE and RMSE values became normal when implementing the loss function training.

4.7.3 The Dilemma of Matrices A and M

While we utilised the SGD learning method to lower both values of equations 3.15 and 3.16 in the previous chapter to reach their local minima, it was difficult to determine if Yao et al. (2018) provided the incorrect operator from equations 18 and 19. We thought the API and mashup vectors derived from TF-IDF values should represent both A and M matrices. As a result, the usage of the vectorised values can cause the program to take a long time to execute the loss function. We have corrected the error by initialising random variables by $a_{ij}, m_{ij} \in \{x \mid 0 \leq x \leq 1 \mid x \in \mathbf{R}\}$ and set the dimensionality of A and M by 40.

4.8 Loss Minimisation Training in Matrices A and M

When the cost function should decrease in every iteration by utilising the SGD method, the MF framework does allow negative values in both matrices A and M. Unlike non-negative matrix factorisation (NMF), this study did not specify what values are allowed in both matrices. NMF has a single rule that only positive values are allowed in both input variables. When we were training both matrices A and M, we randomly set the values lower than 0.05. During training, we have noticed that all values in both matrices were slowly decreasing in each iteration, and it reached negative values in all elements. However, the negative values in both matrices will not affect the scope of this study. As the negative values were present, the standard matrix multiplication cancelled out the negative values and returned the results into positive values. Suppose if we have used both matrices in the NMF loss function defined by Lee and Seung (2001), the experiment will fail. In reality, the MF loss function is not specific, so we assume that it does not belong to any MF subgroups.

4.9 TF-IDF Word Vectorisation

As TF-IDF is commonly used for converting words into numerated vectors, it only outputs the numbers that represent the importance level of a word in a document. It does not directly define the semantics of a word. The values are the frequency of words. By referring to Table 4.1, the document frequency (DF) and inverse document frequency (IDF) values provided meaningless numeral values. TF-IDF does not provide the functionality to hash the corpora into identifiable numbers. If one requires a more accurate approach for recommending APIs for a mashup application, other applications such as deep learning and AI can outperform the simple word calculation of TF-IDF method.

When natural language processing (NLP) applications can perform the task by manipulating the metadata or keywords to produce reliable results, it can process the corpora information in a complicated matter. However, it does not mean that we abandon TF-IDF. Many authors (Basilico & Hofmann, 2004; Bogers & Van den Bosch, 2009; Dong & Shen, 2013; Salter & Antonopoulos, 2006; Tewari et al., 2014; Yao et al., 2013) have published articles that conveyed CF works well with content-based filtering methods and suggests promoting the framework as a hybrid system. Our prototype has implemented CF to calculate the vector-based cosine similarity for a square matrix s_{ij}. However, we did not utilise any content-based filtering in our prototype; it resulted in partially unreliable predictions for the trained R^{prob} matrix.

4.10 Conclusion

In this chapter, we have discussed the experiment set up and the possible difficulties of conducting the procedure for producing the results. We illustrated the graphs and tables to show the results. Additionally, we have reported the details of the charts and tables to convey information about the experiment of this research. For both MAE and RMSE training result in a graph format, the results were consistent with the fitting of the data as it did not underfit or overfit. Although the loss value could be lowered further, the mashup and API invocation worked adequately.

Chapter 5

Conclusion

5.1 Introduction

In this research, we built the matrix factorisation (MF) based recommender systems program that will implicitly increase the probability of an API (application programming interface) invoking other mashups in a matrix. This study has provided answers on how the MF algorithms worked for predicting the probability relationships between an API and every existing web mashup. We will answer the research questions, as defined in Chapter 1. There are limitations for this study and, they have impacted our time used for building the prototype and diagnose the issues encountered during this research. In addition to the drawbacks, we describe possible methods and improvement for future research.

5.2 Answer to Research Questions

1. **How do we integrate context information such as title, description and tags from APIs in order to fill in the unrelated API-mashup relationships?**

   The latent values required for implicit learning was obtained from TF-IDF (term

frequency - inverse document frequency) and cosine similarity calculations. Furthermore, the TF-IDF and cosine similarity calculations are based on the rate of occurrence in every word. With the required latent variables in place, the equations sourced from Yao et al. (2018) were suitable to enhance the recommendation results in addition to slight modifications.

2. **How do existing API-mashup invocations affect the accuracy of the recommendation results?** The machine learning method was used to provide some clues from the existing invocations. Since the stochastic gradient descent (SGD) rule was used for MF, we have observed the increase of relationship between APIs and mashups. Therefore, the results indicated that there is some relationship between unrelated APIs and mashups.

5.3 Limitations

In this study, we have utilised methods to manipulate the API list and the API-mashup invocation dataset to implicitly increase the probability of explicitly unrelated API-mashup invocations. Nevertheless, there are limitations for this research ranging from the manual data preprocessing to experiment results. Because of research constraints, the computing technologies are rapidly evolving, and they may not be guaranteed to work correctly at the first attempt.

- **Selecting the Suitable Quantity of API and Mashup Services**

  Selecting the suitable number of API and mashup services is problematic because it may affect the performance of the program attempting to process and implicitly learn the unrelated APIs. If the number of API and mashup service is small, the results from the experiment can seriously impact the value of the API-mashup invocations. On the other hand, if the number is large, it can decrease the program

performance, and it will take days to complete a section of an algorithm.

- **Data Sparsity Issues**

  Data sparsity problems can result in underfitting of the data. As mentioned in the previous bullet point, the amount of API and mashup services can dictate the reliability of the implicit data following training. It is not easy to guarantee the reliability of the prediction.

- **Dataset Character Mapping Issues**

  Since the dataset can support many characters encoding in Unicode-8, it can cause a problem for the word tokeniser and lemmatiser program to trim and stem words. For example, words containing diacritic symbols such as café can cause the program to stop immediately. This issue will force us to replace diacritic letters by the non-diacritic ones. Another example is the trademark symbol (™) can also halt the process of word processing. Therefore, it can take some time to diagnose and fix this problem by removing the unnecessary symbols from the dataset.

- **API Description Dataset Typos**

  The API description dataset contains some typos which can affect the outcome of the prediction. There are some words of the API names or descriptions were truncated. Truncated words were not done intentionally, but an accident caused by human errors. Therefore, finding the full name of the APIs can sometimes become impossible if the description does not mention it.

- **Scalability Issues**

  The addition of the new APIs can cause problems for the recommendation of web APIs. As the never-ending growth of new APIs are added to the ProgrammableWeb repository, the cold-start issue in recommender systems can affect the

implicit learning of API-mashup invocations.

5.4 Further Research

We have utilised the combined MF technique to recommend APIs based on the implicit invocation of existing mashups. The model still needs some improvements and optimisation to ensure the enhancement of recommender systems. This research has provided us with the fundamentals of the MF algorithms. For future work, we could consider adding more features to the existing MF model to investigate what they can improve for recommender systems. After we have described the limitations of the previous section, we will define possible improvements and future developments for the research.

- **Integration of Neural Networks**

  Including neural networks can be a remarkable idea to enhance the algorithm for improving the results of the recommendation for creating web mashups. By integrating convolutional neural networks (CNN) with the MF cost function (Kim et al., 2017), this allows further investigation of the neural networks and utilise the opportunity to explore its properties in further studies. Also, the latent factor model can be applied to CNN as proposed by Mongia, Jhamb, Chouzenoux and Majumdar (2020) to investigate if it can perform better than collaborative filtering (CF).

- **Tensor Factorisation**

  The tensor factorisation is another possible function to combine with the existing MF framework (Smith, Huang, Sidiropoulos & Karypis, 2018). It allows extra attributes such as time and location to be included in the loss function to ensure the model can learn with the additional features. Besides, it can help to recommend APIs that were based on the timeline or location where users are curious to

investigate what the recent popular items are within the previous timeframe.

- **Corpora to Vectors Processing**

  Since we have used TF-IDF to extract words from the API descriptions dataset, it is possible to use other methods to process every word from the document. The Word2Vec framework is part of the neural architectures that extracts corpora with the self-learning Skip-gram classifier (Arroyo-Fernández et al., 2019). In future research, the Word2Vec architecture will be suitable with the integration of neural networks.

- **Reduction of Data Sparsity**

  As data sparsity is the most common issue with recommender systems, it affected the recommendation performance in the experiment. To combat serious data underfitting, we could combine both CF and content-based filtering methods to reduce the incidence of data sparsity and enhance the likelihood that a possible matching API implicitly invokes with suitable mashups (Yao et al., 2013). Another possible method to attempt is to use the linked open data to calculate the similarity from information mining (Natarajan, Vairavasundaram, Natarajan & Gandomi, 2020).

References

Adomavicius, G. & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering*, *17*(6), 734-749. doi: 10.1109/TKDE.2005.99

Aghdam, M. H., Analoui, M. & Kabiri, P. (2016). Collaborative filtering using non-negative matrix factorisation. *Journal of Information Science*, *43*(4), 567-579. doi: 10.1177/0165551516654354

Ahmad, S., Wasim, S., Gogoi, S. I. S., Srivastava, A. & Farheen, Z. (2019). Qualitative v/s. quantitative research - a summarized review. *Journal of Evidenced Based Medicine and Healthcare 2019*, *6*(43), 2828-2832. doi: 10.18410/jebmh/2019/587

Ajoudanian, S. & Abadeh, M. N. (2019). Recommending human resources to project leaders using a collaborative filtering-based recommender system: Case study of gitHub. *IET Software*, *13*(5), 379-385. doi: 10.1049/iet-sen.2018.5261

Amatriain, X. & Basilico, J. (2016). Past, present, and future of recommender systems: An industry perspective. In *Proceedings of the 10th ACM Conference on Recommender Systems* (p. 211–214). doi: 10.1145/2959100.2959144

Arroyo-Fernández, I., Méndez-Cruz, C.-F., Sierra, G., Torres-Moreno, J.-M. & Sidorov, G. (2019). Unsupervised sentence representations as word information series: Revisiting TF–IDF. *Computer Speech Language*, *56*, 107-129. doi: 10.1016/j.csl.2019.01.005

Basilico, J. & Hofmann, T. (2004). Unifying collaborative and content-based filtering. In *Proceedings of the Twenty-First International Conference on Machine Learning* (p. 9). doi: 10.1145/1015330.1015394

Bobadilla, J., Ortega, F., Hernando, A. & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-Based Systems*, *46*, 109-132. doi: 10.1016/j.knosys.2013.03.012

Bogers, T. & Van den Bosch, A. (2009). Collaborative and content-based filtering for item recommendation on social bookmarking websites. *Journal of Philosophical Logic*, *532*, 9-16.

Cacheda, F., Carneiro, V., Fernández, D. & Formoso, V. (2011). Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Transactions on the Web*, *5*(1). doi: 10.1145/1921591.1921593

Calero Valdez, A., Ziefle, M. & Verbert, K. (2016). HCI for recommender systems: The past, the present and the future. In *Proceedings of the 10th ACM Conference on Recommender Systems* (p. 123–126). doi: 10.1145/2959100.2959158

Cao, B., Liu, X., Rahman, M. D. M., Li, B., Liu, J. & Tang, M. (2017). Integrated content and network-based service clustering and web APIs recommendation for mashup development. *IEEE Transactions on Services Computing*, 1-1. doi: 10.1109/tsc.2017.2686390

Cao, Y., Li, W. & Zheng, D. (2019). A hybrid recommendation approach using LDA and probabilistic matrix factorization. *Cluster Computing*, *22*(4), 8811-8821. doi: 10.1007/s10586-018-1972-y

Cao, Y., Liu, J., Shi, M., Cao, B., Chen, T. & Wen, Y. (2019). Service recommendation based on attentional factorization machine. In *2019 IEEE International Conference on Services Computing (SCC)* (p. 189-196). doi: 10.1109/SCC.2019.00040

Cohen, D., Aharon, M., Koren, Y., Somekh, O. & Nissim, R. (2017). Expediting exploration by attribute-to-feature mapping for cold-start recommendations. In *Proceedings of the Eleventh ACM Conference on Recommender Systems* (p. 184–192). doi: 10.1145/3109859.3109880

Dong, K. & Shen, Y. (2013). A hybrid content-based filtering approach: Recommending microbloggers for web-based communities. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing* (p. 1254-1258). doi: 10.1109/GreenCom-iThings-CPSCom.2013.218

Feng, C., Liang, J., Song, P. & Wang, Z. (2020). A fusion collaborative filtering method for sparse data in recommender systems. *Information Sciences*, *521*, 365-379. doi: 10.1016/j.ins.2020.02.052

Fletcher, K. (2019). Regularizing matrix factorization with implicit user preference embeddings for web api recommendation. In *2019 IEEE International Conference on Services Computing (SCC)* (p. 1-8). doi: 10.1109/SCC.2019.00014

Fogel, P., Hawkins, D. M., Beecher, C., Luta, G. & Young, S. S. (2013). A tale of two matrix factorizations. *The American Statistician*, *67*(4), 207-218. doi: 10.1080/00031305.2013.845607

Gai, L. (2014). Pairwise probabilistic matrix factorization for implicit feedback collaborative filtering. In *Proceedings 2014 IEEE International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)* (p. 181-190). doi: 10.1109/SPAC.2014.6982682

Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S. & Schmidt-Thieme, L. (2010). Learning attribute-to-feature mappings for cold-start recommendations. In *2010 IEEE International Conference on Data Mining* (p. 176-185). doi: 10.1109/ICDM.2010.129

Goldberg, D., Nichols, D., Oki, B. M. & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, *35*(12), 61-70. doi: 10.1145/138859.138867

Guo, L., Ma, J., Chen, Z.-M. & Jiang, H.-R. (2014). Incorporating item relations for social recommendation. *Chinese Journal of Computers*, *37*(1).

Retrieved from http://cjc.ict.ac.cn/online/onlinepaper/gl-201411595219.pdf

Guo, L., Wen, Y.-F. & Wang, X.-H. (2018). Exploiting pre-trained network embeddings for recommendations in social networks. *Journal of Computer Science and Technology*, *33*(4), 682-696. doi: 10.1007/s11390-018-1849-9

Hao, Y., Fan, Y., Tan, W. & Zhang, J. (2017). Service recommendation based on targeted reconstruction of service descriptions. In *2017 IEEE International Conference on Web Services (ICWS)* (p. 285-292). doi: 10.1109/icws.2017.44

Hu, Y., Koren, Y. & Volinsky, C. (2008). Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining* (p. 263–272). doi: 10.1109/ICDM.2008.22

Hubert, L., Meulman, J. & Heiser, W. (2000). Two purposes for matrix factorization: A historical appraisal. *SIAM Review*, *42*(1), 68-82. Retrieved from http://www.jstor.org/stable/2653377

Hussain, S., Keung, J., Khan, A. A., Ahmad, A., Cuomo, S., Piccialli, F., ... Akhunzada, A. (2018). Implications of deep learning for the automation of design patterns organization. *Journal of Parallel and Distributed Computing*, *117*, 256-266. doi: 10.1016/j.jpdc.2017.06.022

Iwahama, K., Hijikata, Y. & Nishida, S. (2004). Content-based filtering system for music data. In *2004 International Symposium on Applications and the Internet Workshops. 2004 Workshops.* (p. 480-487). doi: 10.1109/SAINTW.2004.1268677

Jain, A., Liu, X. & Yu, Q. (2015). Aggregating functionality, use history, and popularity of APIs to recommend mashup creation. In *13th International Conference, ICSOC 2015* (p. 188-202). doi: 10.1007/978-3-662-48616-0_12

Jiang, S., Li, K. & Xu, R. Y. D. (2019). Relative pairwise relationship constrained non-negative matrix factorisation. *IEEE Transactions on Knowledge and Data Engineering*, *31*(8), 1595-1609. doi: 10.1109/TKDE.2018.2859223

Kim, D., Park, C., Oh, J. & Yu, H. (2017). Deep hybrid recommender systems via exploiting document context and statistics of items. *Information Sciences*, *417*, 72-87. doi: 10.1016/j.ins.2017.06.026

Kim, K.-J. & Ahn, H. (2011). Collaborative filtering with a user-item matrix reduction technique. *International Journal of Electronic Commerce*, *16*(1), 107-128. doi: 10.2753/JEC1086-4415160104

Kluver, D. & Konstan, J. A. (2014). Evaluating recommender behavior for new users. In *Proceedings of the 8th ACM Conference on Recommender Systems* (p. 121–128). doi: 10.1145/2645710.2645742

Koren, Y., Bell, R. & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer Society*.

Kunaver, M. & Požrl, T. (2017). Diversity in recommender systems – a survey. *Knowledge-Based Systems*, *123*, 154-162. doi: 10.1016/j.knosys.2017.02.009

Lee, D. D. & Seung, H. S. (2001). Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems 13* (p. 556-562). Retrieved from http://papers.nips.cc/paper/1861-algorithms-for-non-negative-matrix-factorization.pdf

Liu, H.-F., Jing, L.-P. & Yu, J. (2018). Survey of matrix factorization based recommendation methods by integrating social information. *Journal of Science*, *29*(2), 340-362. doi: 10.13328/j.cnki.jos.005391

Liu, S.-D. & Meng, X.-W. (2015). Recommender systems in location-based social networks. *Chinese Journal of Computers*, *38*(2). Retrieved from http://cjc.ict.ac.cn/online/onlinepaper/lsd-201523212539.pdf

Looi, W., Dhaliwal, M., Alhajj, R. & Rokne, J. (2019). Recommender system for items in dota 2. *IEEE Transactions on Games*, *11*(4), 396-404. doi: 10.1109/TG.2018.2844121

Ma, H., Zhou, D., Liu, C., Lyu, M. R. & King, I. (2011). Recommender systems with social regularization. In *Proceedings of the Fourth ACM International Conference on Web Search and Data Mining* (p. 287–296). doi: 10.1145/1935826.1935877

Madani, Y., Erritali, M., Bengourram, J. & Sailhan, F. (2019). Social collaborative filtering approach for recommending courses in an e-learning platform. *Procedia Computer Science*, *151*, 1164-1169. doi: 10.1016/j.procs.2019.04.166

Mnih, A. & Salakhutdinov, R. R. (2008). Probabilistic matrix factorization. In *Advances in Neural Information Processing Systems 20* (p. 1257-1264). Retrieved from http://papers.nips.cc/paper/3208-probabilistic-matrix-factorization.pdf

Mongia, A., Jhamb, N., Chouzenoux, E. & Majumdar, A. (2020). Deep latent factor model for collaborative filtering. *Signal Processing*, *169*, 107366. doi: 10.1016/j.sigpro.2019.107366

Mu, Y., Xiao, N., Tang, R., Luo, L. & Yin, X. (2019). An efficient similarity measure for collaborative filtering. *Procedia Computer Science*, *147*, 416-421. doi: 10.1016/j.procs.2019.01.258

Natarajan, S., Vairavasundaram, S., Natarajan, S. & Gandomi, A. H. (2020). Resolving data sparsity and cold start problem in collaborative filtering recommender system using linked open data. *Expert Systems with Applications*, *149*, 113248. doi: 10.1016/j.eswa.2020.113248

Pal, A., Parhi, P. & Aggarwal, M. (2017). An improved content based collaborative filtering algorithm for movie recommendations. In *2017 Tenth International Conference on Contemporary Computing (IC3)* (p. 1-3). doi: 10.1109/IC3.2017.8284357

Pera, M. S. & Ng, Y.-K. (2018). Recommending books to be exchanged online in the absence of wish lists. *Journal of the Association for Information Science and Technology*, *69*(4), 541-552. doi: 10.1002/asi.23978

Porter, M. F. (1980). An algorithm for suffix stripping. *Program*, *14*(3), 130-137. doi: 10.1108/eb046814

Punyabukkana, P. (2017). Teaching research methods for computer science students using active learning approach. In *2017 IEEE 6th International Conference on Teaching, Assessment, and Learning for Engineering (TALE)* (p. 256-260). doi: 10.1109/TALE.2017.8252343

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. & Riedl, J. (1994). Grouplens:

An open architecture for collaborative filtering of netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work* (p. 175-186). doi: 10.1145/192844.192905

Resnick, P. & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, *40*(3), 56–58. doi: 10.1145/245108.245121

Salter, J. & Antonopoulos, N. (2006). Cinemascreen recommender agent: Combining collaborative and content-based filtering. *IEEE Intelligent Systems*, *21*(1), 35-41. doi: 10.1109/MIS.2006.4

Schein, A. I., Popescul, A., Ungar, L. H. & Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (p. 253–260). doi: 10.1145/564376.564421

Sedhain, S., Menon, A. K., Sanner, S., Xie, L. & Braziunas, D. (2017). Low-rank linear cold-start recommendation from social data. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)* (p. 1502-1508). Retrieved from `http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14828`

Sedhain, S., Sanner, S., Braziunas, D., Xie, L. & Christensen, J. (2014). Social collaborative filtering for cold-start recommendations. In *Proceedings of the 8th ACM Conference on Recommender Systems* (p. 345–348). doi: 10.1145/2645710.2645772

Shakirova, E. (2017). Collaborative filtering for music recommender system. In *2017 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)* (p. 548-550). doi: 10.1109/EIConRus.2017.7910613

Shi, Y., Larson, M. & Hanjalic, A. (2014). Collaborative filtering beyond the user-item matrix: A survey of the state of the art and future challenges. *ACM Computing Surveys*, *47*(1). doi: 10.1145/2556270

Singh, J. & Gupta, V. (2019). A novel unsupervised corpus-based stemming technique using lexicon and corpus statistics. *Knowledge-Based Systems*, *180*, 147-162. doi: 10.1016/j.knosys.2019.05.025

Smith, B. & Linden, G. (2017). Two decades of recommender systems at Amazon.com. *IEEE Internet Computing*, *21*(3), 12-18. doi: 10.1109/MIC.2017.72

Smith, S., Huang, K., Sidiropoulos, N. D. & Karypis, G. (2018). Streaming tensor factorization for infinite data sources. In *Proceedings of the 2018 SIAM International Conference on Data Mining* (p. 81-89). doi: 10.1137/1.9781611975321.10

Sohrabi, B., Toloo, M., Moeini, A. & Nalchigar, S. (2015). Evaluation of recommender systems: A multi-criteria decision making approach. *Iranian Journal of Management Studies (IJMS)*, *8*(4), 589-605. doi: 10.22059/ijms.2015.55003

Su, X. & Khoshgoftaar, T. M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 1-9. doi: 10.1155/2009/421425

Tang, J., Hu, X. & Liu, H. (2013). Social recommendation: A review. *Social Network Analysis and Mining*, *3*(4), 1113-1133. doi: 10.1007/s13278-013-0141-9

Tewari, A. S., Kumar, A. & Barman, A. G. (2014). Book recommendation system based

on combine features of content based filtering, collaborative filtering and association rule mining. In *2014 IEEE International Advance Computing Conference (IACC)* (p. 500-503). doi: 10.1109/IAdCC.2014.6779375

Turnip, R., Nurjanah, D. & Kusumo, D. S. (2017). Hybrid recommender system for learning material using content-based filtering and collaborative filtering with good learners' rating. In *2017 IEEE Conference on e-Learning, e-Management and e-Services (IC3e)* (p. 61-66). doi: 10.1109/IC3e.2017.8409239

Verma, S. K., Mittal, N. & Agarwal, B. (2013). Hybrid recommender system based on fuzzy clustering and collaborative filtering. In *2013 4th International Conference on Computer and Communication Technology (ICCCT)* (p. 116-120). doi: 10 .1109/ICCCT.2013.6749613

Wang, G., Jiang, J., Wang, H.-R. & Yang, S.-L. (2019). Study of group recommendation based on probabilistic matrix factorization. *Chinese Journal of Computers*, *42*(1). doi: 10.11897/SP.J.1016.2019.00098

Wang, H., Tao, Y., Yu, Q., Lin, X. & Hong, T. (2018). Incorporating both qualitative and quantitative preferences for service recommendation. *Journal of Parallel and Distributed Computing*, *114*, 46-69. doi: 10.1016/j.jpdc.2017.12.005

Wang, X., Lu, W., Ester, M., Wang, C. & Chen, C. (2016). Social recommendation with strong and weak ties. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (p. 5–14). doi: 10.1145/ 2983323.2983701

Wang, X., Zhang, W., Yan, J., Yuan, X. & Zha, H. (2018). On the flexibility of block coordinate descent for large-scale optimization. *Neurocomputing*, *272*, 471-480. doi: 10.1016/j.neucom.2017.07.024

Xiang, R., Neville, J. & Rogati, M. (2010). Modeling relationship strength in online social networks. In *Proceedings of the 19th International Conference on World Wide Web* (p. 981-990). doi: 10.1145/1772690.1772790

Xie, B., Han, P., Yang, F., Shen, R.-M., Zeng, H.-J. & Chen, Z. (2007). DCFLA: A distributed collaborative-filtering neighbor-locating algorithm. *Information Sciences*, *177*(6), 1349 - 1363. doi: 10.1016/j.ins.2006.09.005

Xiong, R., Wang, J., Zhang, N. & Ma, Y. (2018). Deep hybrid collaborative filtering for web service recommendation. *Expert Systems with Applications*, *110*, 191-205. doi: 10.1016/j.eswa.2018.05.039

Yang, X., Guo, Y., Liu, Y. & Steck, H. (2014). A survey of collaborative filtering based social recommender systems. *Computer Communications*, *41*, 1-10. doi: 10.1016/j.comcom.2013.06.009

Yang, X., Steck, H. & Liu, Y. (2012). Circle-based recommendation in online social networks. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (p. 1267–1275). doi: 10.1145/2339530 .2339728

Yao, L., Sheng, Q. Z., Ngu, A. H. H., Yu, J. & Segev, A. (2015). Unified collaborative and content-based web service recommendation. *IEEE Transactions on Services Computing*, *8*(3), 453-466. doi: 10.1109/TSC.2014.2355842

Yao, L., Sheng, Q. Z., Segev, A. & Yu, J. (2013). Recommending web services via

combining collaborative filtering with content-based features. In *2013 IEEE 20th International Conference on Web Services* (p. 42-49). doi: 10.1109/ICWS.2013.16

Yao, L., Wang, X., Sheng, Q. Z., Benatallah, B. & Huang, C. (2018). Mashup recommendation by regularizing matrix factorization with API co-invocations. *IEEE Transactions on Services Computing*, 1-1. doi: 10.1109/tsc.2018.2803171

Yu, Z., Wong, R. K. & Chi, C.-H. (2017). Efficient role mining for context-aware service recommendation using a high-performance cluster. *IEEE Transactions on Services Computing*, *10*(6), 914-926. doi: 10.1109/tsc.2015.2485988

Zang, Y., An, Y. & Hu, X. T. (2014). Automatically recommending healthy living programs to patients with chronic diseases through hybrid content-based and collaborative filtering. In *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (p. 578-582). doi: 10.1109/BIBM.2014.6999224

Zetlaoui, M., Feinberg, M., Verger, P. & Clémençon, S. (2011). Extraction of food consumption systems by nonnegative matrix factorization (NMF) for the assessment of food choices. *Biometrics*, *67*(4), 1647-1658. doi: 10.1111/j.1541-0420.2011.01588.x

Zhang, R., Liu, Q., Chun-Gui, Wei, J. & Huiyi-Ma. (2014). Collaborative filtering for recommender systems. In *2014 Second International Conference on Advanced Cloud and Big Data* (p. 301-308). doi: 10.1109/CBD.2014.47

Zhang, S., Wang, W., Ford, J. & Makedon, F. (2006). Learning from incomplete ratings using non-negative matrix factorization. In *Proceedings of the 2006 SIAM International Conference on Data Mining* (p. 549-553). doi: 10.1137/1.9781611972764.58

Zhang, W., Liu, F., Xu, D. & Jiang, L. (2019). Recommendation system in social networks with topical attention and probabilistic matrix factorization. *PLoS One*, *14*(10), 1-18. doi: 10.1371/journal.pone.0223967

Zhang, Y., Meng, K., Kong, W. & Dong, Z. Y. (2019). Collaborative filtering-based electricity plan recommender system. *IEEE Transactions on Industrial Informatics*, *15*(3), 1393-1404. doi: 10.1109/TII.2018.2856842

Zhao, F., Yan, F., Jin, H., Yang, L. T. & Yu, C. (2017). Personalized mobile searching approach based on combining content-based filtering and collaborative filtering. *IEEE Systems Journal*, *11*(1), 324-332. doi: 10.1109/JSYST.2015.2472996

Zhong, Y. & Fan, Y. (2017). Extracting relevant terms from mashup descriptions for service recommendation. *Tsinghua Science and Technology*, *22*(3), 293-302. doi: 10.23919/TST.2017.7914201

Appendix A

Glossary and Abbreviations

Algorithm: A sequence of ordered instructions to achieve its computing solutions.

Application programming interface (API): A computing interface that provides a set of instructions to run an application by another, equivalent to a restaurant menu.

Artificial intelligence (AI): A computing method which resembles a human brain that can execute tasks and learn artificially without needing manual intervention.

Central processing unit (CPU): A computer hardware is located in the heart of a motherboard that executes commands from a program.

Coefficient: A number, known or unknown that is to be multiplied.

Collaborative filtering (CF): A filtering method that outputs results based on the users' tastes.

Comma-separated values (CSV): The file format that stores text separated by a comma.

Content-based filtering: A filtering method that provides outcomes based on attributes such as name, age and tags.

Convergence: The end point (minimum or maximum) following the implicit learning of a cost function.

Convolutional neural networks (CNN): A class of neural networks that utilises the complex neurons based on deep learning motivated by the biological nervous system.

Corpus: The text of a document.

DDR4: Double data rate 4

Derivative: A reduced mathematical function produced following the calculus differentiation of an original one.

DF: Document frequency

Equation: A mathematical expression that calculates the output value equal to the opposite side.

Function: A set of mathematical instructions, usually equations that manipulates the input values and provides a resulting outcome.

GB: Gigabytes

GDDR5: Graphics double data rate 5

HDD: Hard disk drive

IDF: Inverse document frequency

***k*-nearest neighbours (*k*NN):** The algorithm that finds a finite quantity of nearest items based on the k number of values.

Latent variable: A variable that is not explicitly known.

MAE: Mean absolute error

Mashup: A computer application that relies on external software in order to achieve the purpose of its function.

Matrix: A rectangular array that stores a set of numbers.

Matrix factorisation (MF): A method of computation that utilises the matrix multiplication method.

Metadata: Attributes of a file.

Natural language processing (NLP): A process part of artificial intelligence that

processes human languages by speech and written words.

NMF: Non-negative matrix factorisation

Overfitting: Data that corresponds very close to a regression curve.

PMF: Probabilistic matrix factorisation

PPMF: Pairwise probabilistic matrix factorisation

Random-access memory (RAM): A computer memory that is installed on a motherboard that stores current temporary data of a running application.

Recommender systems (RS): The framework for recommending items to users.

RMSE: Root mean square error

RPR-NMF: Relative pairwise relationship - non-negative matrix factorisation

SDK: Software development kit

Stochastic gradient descent (SGD): A machine learning method that lowers the value until it reaches its convergence point.

SVD: Singular value decomposition

TB: Terabytes

TF: Term frequency

TF-IDF: Term frequency - inverse document frequency

Tokenise: The procedure for dividing all words and symbols from a large text into a list of separated words and symbols.

Underfitting: Data that are scattered outside a regression curve.

Variable: A computing or programming storage location for storing any data.

WNMF: Weighted non-negative matrix factorisation

Appendix B

Prototype Implementation

The prototype for this research is in cloud storage, and the link for the zip file is:

https://www.dropbox.com/sh/8xbvc9zjvcpoa43/AAA1ZdHgq0qssjZ9dDBwG9y8a?dl=0