COMMUNITY RECOMMENDATION IN SOCIAL NETWORKS WITH SPARSE DATA

A Thesis

Submitted to the Faculty

of

Purdue University

by

Emad Rahmaniazad

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

December 2020

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

ACKNOWLEDGMENTS

I would like to thank all of my advisors, Dr. Ali Jafari, Dr. Brian King, and Dr. Paul Salama, for their mentorship, and the graduate programs coordinator, Sherrie Tucker, for her consistent help and support. Also, I wanted to thank all the Course Networking's employees and the CyberLab team for providing the opportunity for me to complete my thesis. Last but not least, a special thanks to my family for their support.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Rahmaniazad, Emad. M.S.E.C.E., Purdue University, December 2020. Community Recommendation in Social Networks with Sparse Data. Major Professors: Brian King and Ali Jafari.

Recommender systems are widely used in many domains. In this work, the importance of a recommender system in an online learning platform is discussed. After explaining the concept of adding an intelligent agent to online education systems, some features of the Course Networking (CN) website are demonstrated. Finally, the relation between CN, the intelligent agent (Rumi), and the recommender system is presented. Along with the argument of three different approaches for building a community recommendation system. The result shows that the Neighboring Collaborative Filtering (NCF) outperforms both the transfer learning method and the Continuous bag-of-words approach. The NCF algorithm has a general format with two various implementations that can be used for other recommendations, such as course, skill, major, and book recommendations.

1. INTRODUCTION

Recommender systems are methods utilized in a variety of areas to give users recommendations based on their preferences. Over the last ten years, with online information growth, designing and evaluating such systems has been necessary to solve information overload problems [1]. As a result, recommendation systems have been widely implemented in many internet activities. Some examples worth mentioning are e-commerce, web pages, censorship sectors, and other sectors such as news, tourist information, and e-learning [2].

In this work, the focus is on using recommender systems in the learning management system (LMS), which is an e-learning platform. A learning management system is a software application or web-based technology used to design, perform, and evaluate a specific learning process [3]. CourseNetworking (CN) is an instance of such an LMS where users from all around the world can join and share their information like academic achievements, work background, skills, and join different communities related to their interests.

Online communities play an essential role in enhancing the degree of user engagement or the number of interactions. The number of user interactions positively correlates with user satisfaction and hence, the system's success. The more members get involved, the more productive that network will be. However, it is not easy to have an acceptable user engagement rate on a social platform. For example, analyzing the CN data shows that the majority number of CN users have not participated in any communities. That is, there might be a group of users who do not have the time to find relevant communities using the search tools. In this case, we need a method to increase the engagement of those inactive users or to improve users' decision-making process.

Applying a community recommender system might be an effective solution to this problem. Community recommender systems help users to join communities. Such systems can also build a lively environment for the social network's existing communities to attract more members, resulting in more interactions. Moreover, having users join communities makes it easier to give them recommendations in the future. Before exploring the details of implementing a community recommender system in CN, it is needed to learn more about recommender systems.

Recommender systems can take different approaches or techniques. Some of primary Recommendation system approaches are proposed in learning objects are as follow [2]:

- Content-based System (CBS)
- Collaborative Filtering System (CFS)
- Demographic-based System (DBS)
- Hybrid Recommender System (HRS)

In CBS [2], the recommender engine gives recommendations by finding items similar to what the user is known to like. In the CFS [2] approach, the engine tries to find people who have similar consumption patterns to the user and then recommends items that those people might like. A DBS [2] makes decisions by categorizing users based on the demographic group they belong to, such as income, age, and learning level. Finally, in HRS [2], two or more recommender techniques are combined to gain better performance.

This work aims to develop a recommender system to provide personalized community recommendations for CN members. This system is implemented under the concept of an Intelligent Agent, known as Rumi. This agent has to first gain the user's trust by providing them useful information. Then the chance of acceptance will increase if Rumi recommends communities to the user. Whereas recommending things to users from the system without involving an agent will be static and less amusing.

Three techniques have been used to build the community recommendation engine; collaborative filtering (CF), Word2Vec, a content-based approach, and transfer learning, a state-of-the-art hybrid approach. Each of which has been modified based on the CN dataset. Before applying any of the modified models, the dataset was cleaned and prepared. In other words, each method requires a unique data preprocessing pipeline to remove inconsistent data and a translation to convert the data to inputs and outputs of the models. In the end, all approaches have been compared.

The next chapter discusses the previous research related to this work. More information about CN and its functionalities are provided in Chapter 3. Chapter 4 explains the reasons for implementing Rumi in the CN and the services he provides. The most critical chapter of this work is Chapter 5, where the three approaches for building a community recommendations are argued. This chapter also illustrates some of the designs related to how Rumi will recommend communities to the user. Finally, the last chapter includes a conclusion of this work and the ways to improve it.

2. RELATED WORK

2.1 Recommender Systems

There are many studies related to Recommender Systems and language understanding. In this section, some of the existing related work is reviewed through three topics. Each of these research works has been selected to be considered based on several criteria, such as their proposed method's performance, their dataset similarity with CN dataset, time and space consumption, and other limitations and downsides.

Determining the most suitable recommender system is not trivial. There are three main evaluation methods to measure recommender systems quality: user studies, online evaluations, and offline evaluations [4]. Additionally, an improper evaluation metric can lead to selecting an inappropriate algorithm for the task of interest. The choice of evaluation metric is made by the recommender systems' core tasks—the prediction task, the recommendation task, and the utility maximization task [5].

Online evaluations measure user satisfaction implicitly [6]. In online evaluations, recommendations are shown to real users of the system during their session [4]. Then, the recommender system tracks how often a user admits a recommendation. Acceptance is frequently measured by click-through rate (CTR) [5]. On the other hand, offline evaluations use pre-compiled datasets from which some information has been removed. Subsequently, the recommender algorithms are examined on their ability to recommend the missing information [5] [6] .

On LinkedIn.com, there is a feature called "Skills and Expertise," where users can tag themselves with topics to represent their expertise areas. The goal in [7] is to form a large enough list of skills and expertise that members might pick among to add to their profile. A topic extraction pipeline is proposed in [7], which includes creating a folksonomy of skills and expertise. The folksonomy is constructed from the members' data by taking three steps: discovery, disambiguation, deduplication. In the discovery phase, [7] extracts phrases from the profiles. After finding an initial set of skill phrases, it is needed to disambiguate those phrases that possessed multiple meanings depending on their context. Next, the proposed pipeline removes skill phrases that are semantically duplicate of each other. Finally, the most likely skills are offered to users based on their profiles and the list of skills [7].

2.1.1 Collaborative Filtering

One online recommendation algorithm is Item-to-Item CF, developed by Amazon.com. The existing recommendation algorithms cannot scale to Amazon.com's massive dataset of customers and products; therefore, Amazon.com builds its own algorithm [8]. In this approach, for each item purchased by the user, Amazon.com makes a neighborhood of related items and recommends the most similar ones to the user. The similarity metric between two items is calculated based on how often customers tend to buy them together. The advantage of this method is that it can react immediately to the user's interaction. In contrast, three common recommending approaches, traditional CF, cluster models, and search-based methods, cannot find a set of suitable recommendations in less than a second. One of the downsides of the item-to-time algorithm is that many product pairs have no common customers; consequently, the approach is inefficient in terms of processing time and memory usage [8].

An item-based CF algorithm is proposed in [9], in which a user receives a recommendation based on the items the user has rated. It computes the similarity between two items by isolating the users who have rated both of those items, then applies Pearson-r correlation and adjusted cosine techniques to determine the similarity. Finding the k most similar items to a target item, [9] generates predictions. The approach proposed by [9] fails to recommend new items to the users who have not rated any items, where cold start happens.

The approach proposed by [10], Combinational Collaborative Filtering (CCF), is a hybrid method that formulates personalized community recommendations on Orkut dataset. This algorithm fuses the user-based CF method (bag of users) with the words describing a community (bag of words) to solve the database's sparsity problem. One of the downsides of the proposed algorithm is computation complexity, which results from utilizing Gibbs Sampling and Expectation-Maximization (EM) techniques. To solve this problem, computation is distributed among several machines by using the parallelization method. Parallel computing speeds up the model training and improves the quality of recommendations [10].

Instead of recommending items to an individual user, [11] concentrates on making recommendations for a group of people in a community by defining a metric, Community Similarity Degree (CSD). CSD quantifies the inner connection density of community members by their common interests. This metric has a lower computation complexity compared to the conventional approaches, such as cosine similarity. The CSD metric's effectiveness is evaluated by utilizing a web crawler that follows the Breadth-First Search (BFS) approach to collect users' information from Facebook. Then, users are sorted into four different types of communities. The authors' experiments show that recommending communities to users based on the CSD metric has the best accuracy in all community types [11].

2.1.2 Word2vec Model

Two model architectures, a continuous bag of words (CBOW) and Skip-Gram are proposed by [12] for computing vector representations of words. CBOW maximizes the target word's probability by looking at the context, which can be problematic for rare words. On the other hand, given the distributed representation of the input word, the Skip-Gram model is designed to predict the context. The experiments show that the Skip-Gram model works well with small datasets and outputs a better representation for even rare words or phrases. However, the CBOW model exhibits better performance on frequent words and runs faster than the Skip-Gram model. Comparing the quality of vector representations in terms of syntactic and semantic similarities shows that both the CBOW and Skip-Gram models outperform the popular neural network models (both feedforward and recurrent). Also, they have less time complexity and better accuracy in larger datasets [12].

Mikolov et al. present several extensions in [13] to improve both the quality of the vectors and the training speed of the Skip-Gram model. They improve the algorithm's speedup by subsampling frequent words. This also helps to learn more regular word representations. Word embedding techniques [13] use the skip-gram model to generate word representations for a large corpus. According to [13], one of the major pitfalls when performing word embeddings is the high computational cost of performing softmax on the output. Negative Sampling in [13] outperforms the hierarchical softmax, which uses the Hoffman tree to present the word frequency.

Negative Sampling [13] uses Noise Contrastive Estimation (NCE) [14] to improve the computational efficiency of the skip-gram model by updating the weights only for a subset of the nodes, not all the outputs. The negative samples are chosen randomly based on the number of their occurrences in the corpus. The authors also display the success of their algorithm in learning idiomatic phrases in [13].

The choice of hidden units' activation functions is important in designing an optimal neural network for a given dataset. Utilizing different activation functions in neural networks produces various results. Five different activation functions, Bi-Polar Sigmoid, Uni-Polar Sigmoid, Tanh, Conic Section, and Radial Bases Function (RBF), are used to a multi-layered perceptron (MLP) neural network architecture. Comparing the results show that Tanh has the most accuracy and outperforms others in most cases; however, this still one might not obtain the same result on every dataset. The best activation function needs to be selected based on the dataset and the problem [15].

2.1.3 Transfer Learning

Although neural networks are proven to be robust methods for supervised learning tasks, their effectiveness is limited due to insufficient data. In most cases, collecting new data and labeling them to feed into the network are expensive or time-consuming. In recent years, transfer learning proposed a reliable solution to this problem. In transfer learning, instead of initializing the parameters randomly at the beginning of the training, a pre-trained model on another dataset can be used. Therefore, there is no need to hold the assumption that training and test data are needed to have the same distribution [16].

Most current research focuses on the deployment of transfer learning in deep neural networks, called deep transfer learning. Deep transfer learning is classified into four categories in [17] as follows: instances-based deep transfer learning, mapping-based deep transfer learning, network-based deep transfer learning, and adversarial-based deep transfer learning. In most real-world applications, a hybrid method is utilized to achieve better performance.

The transfer learning in NLP tasks can be divided into pre-training and fine-tuning tasks. There are many standard language models to tackle NLP tasks. One of the significant limitations of standard language models is their unidirectional property, where only a left-to-right or right-to-left architecture can be used during pre-training. In contrast, Bidirectional Encoder Representations from Transformers (BERT) model developed by Google.com [18] applies masked language models (MLM) to enable pre-trained deep bidirectional representations. Mask language model learns to understand the relationship between words by randomly choosing some of the tokens from the input and predicting the masked word based on its context. The pre-training BERT model on a large corpus helps the model to get a deeper understanding of how language works. Besides the masked language models, BERT is also pre-trained on the Next Sentence Prediction task to understand the relationship between sentences [18].

2.2 Intelligent Agent

The lack of an agent in learning systems has been argued in [19]. The author in [19] conceptualized three different intelligent agents and claims that learning systems can benefit from these agents. In addition, [20] states that implementing agents can make the dynamism in the learning process more powerful. The differences between conventional software and an agent are discussed in [21]. Some of the properties for these differences are as follows:

- Autonomous: The agent can make decisions based on its reasoning and without any implicit permission. Whereas conventional software waits for commands.
- Reactive: The agent can find out about any changes in the system and adjusts its functionality before reacting. In contrast, conventional software is programmed to take specific actions for predefined changes.
- Trustworthy: The agent listens to its owner and does not cross the line to gain trust.
- Personalized: The agent learns over time and is taught what to do in every condition. On the other hand, conventional software has limitations for being controlled by the user.
- Social behavior: The agent interacts with members to help them reach their goals. This communication mimics real-world conversations. However, conventional software mostly takes commands.

A graduate student, Seyed Mahmood Hosseini Asanjan, has proposed an online personal assistant that integrates with CourseNetworking and can be integrated with other institutions' learning system through a set of RESTful APIs [22]. The implementation of this software took place at the research and development laboratory of CN, CyberLab. The software supports a model-view-controller (MVC) architecture and is written in PHP 7.1 and uses Laravel framework. It runs on the Amazon Web Services (AWS) cloud platform and has a MongoDB database.

The agent collects data from users and adjusts its performance by utilizing machine learning techniques. The software also has two engines to decide when to show the agent and what content to respond. One of which is the Reasoning engine that computes the required response for any of the API requests. The other engine is the Priority engine, which finds the feature with higher priority among all the features. These features include announcements, friend recommendations, and job recommendations. Priority engine has a 12 hours cycle before previewing the agent again. It sorts the features based on their importance and in this cycle.

3. COURSE NETWORKING

3.1 Learning Management System

A conventional Learning Management System (LMS) focuses on managing the course-related objects, stated in [23]. The white paper discusses two of the flaws of these systems. The first flaw is that a conventional LMS aims to connect a limited number of users within a classroom or broader scope; it only links people on one campus. Even if other institutions license the same platform, there is no connection between members of different institutions.

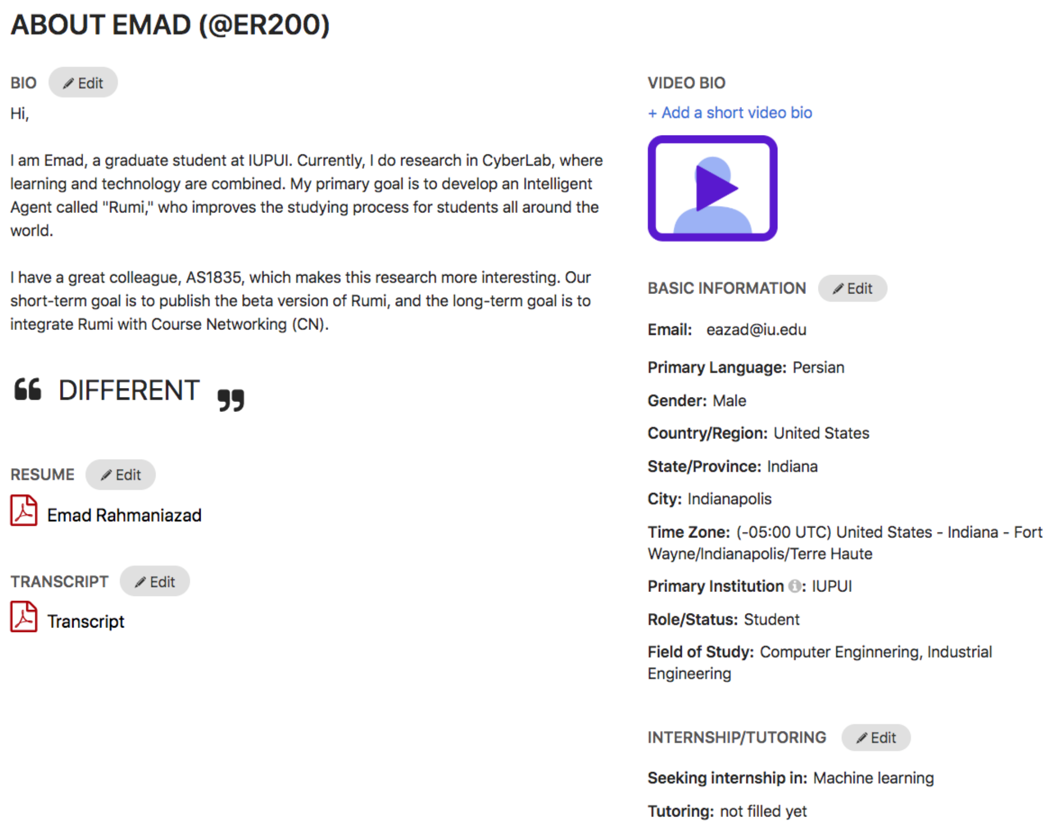
Another issue with a conventional LMS is that they pay little attention to how user-friendly their system is [23]. For instance, several clicks are needed for reaching necessities on the system. Therefore, a manual or support team is required to give specific instructions.

CourseNetworking (CN) is an instance of an LMS which not only manages the courses but also solves the two problems mentioned. CN has removed the boundaries between institutions, and students from different campuses can message each other. This interaction can be the beginning of a new network for remote collaboration. To overcome the second issue, CN designs user interfaces (UIs) with higher usability, and consequently, fewer clicks and instructions are required for interacting with the system.

In addition, CN offers two other tools; a digital resume and a social network. The former tool is called an ePortfolio, where users can express themselves through various sections. The latter feature connects instructors, students, and employers from all around the world.

3.2 ePortfolio

CN, like many job oriented websites, has its ePortfolio where one can express their achievements, educational background, and expertise. CN's ePortfolio has many sections, but this project only focuses on some of them. One of which is the About Section, shown in Figure 3.1. In this part, users can write a short bio, add crucial documents, write a tagline or quote, and input the basic information such as their Field of Study.


ABOUT EMAD (@ER200)

BIO Edit

Hi,

I am Emad, a graduate student at IUPUI. Currently, I do research in CyberLab, where learning and technology are combined. My primary goal is to develop an Intelligent Agent called "Rumi," who improves the studying process for students all around the world.

I have a great colleague, AS1835, which makes this research more interesting. Our short-term goal is to publish the beta version of Rumi, and the long-term goal is to integrate Rumi with Course Networking (CN).

“ DIFFERENT ”

RESUME Edit

Emad Rahmaniazad

TRANSCRIPT Edit

Transcript

VIDEO BIO

+ Add a short video bio

BASIC INFORMATION Edit

Email: eazad@iu.edu

Primary Language: Persian

Gender: Male

Country/Region: United States

State/Province: Indiana

City: Indianapolis

Time Zone: (-05:00 UTC) United States - Indiana - Fort Wayne/Indianapolis/Terre Haute

Primary Institution: IUPUI

Role/Status: Student

Field of Study: Computer Enginnering, Industrial Engineering

INTERNSHIP/TUTORING Edit

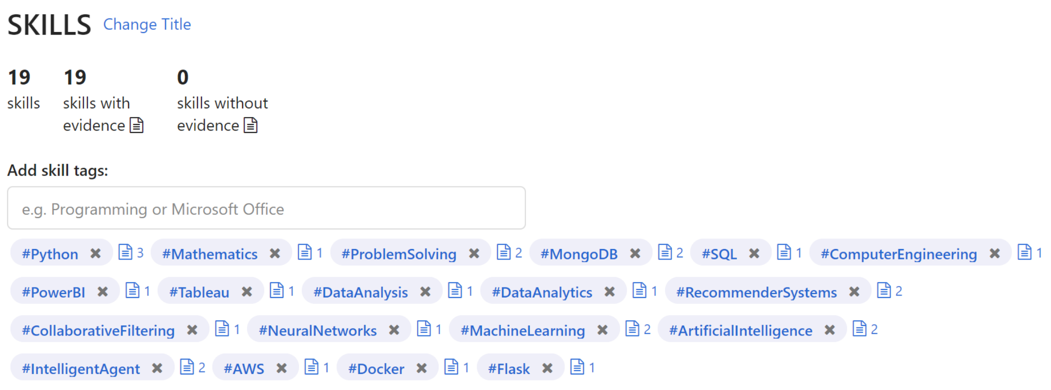
Seeking internship in: Machine learning

Tutoring: not filled yet

Fig. 3.1.: About Section

The second division is the Skills Section that contains the member's skills (Figure 3.2). This is a popular part in most portfolios because it is easier for both reader and writer to identify skill tags or phrases rather than find out about them when the word or phrase is embedded in the text. In fact, what makes CN's version unique is the ability to showcase previous work and link the showcase to skills in the Skills Section. These showcases are displayed in the Showcase Section.

An example of these showcases, along with skills demonstrated in work, is shown in Figure 3.3. Users can explain their work in detail and attach documents, images, and videos. More so, they can modify the visibility and create shareable links to send it to employers. As a result, employers are more attracted if an accomplishment follows a skill with a detailed explanation.


SKILLS Change Title

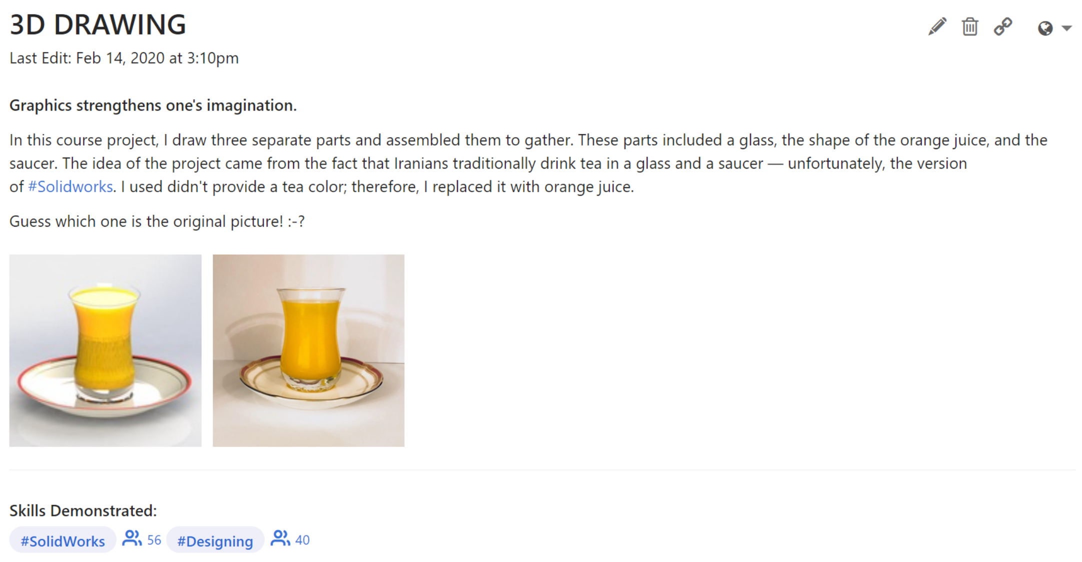
19 skills | 19 skills with evidence | 0 skills without evidence

Add skill tags:

e.g. Programming or Microsoft Office

#Python 3 #Mathematics 1 #ProblemSolving 2 #MongoDB 2 #SQL 1 #ComputerEngineering 1
#PowerBI 1 #Tableau 1 #DataAnalysis 1 #DataAnalytics 1 #RecommenderSystems 2
#CollaborativeFiltering 1 #NeuralNetworks 1 #MachineLearning 2 #ArtificialIntelligence 2
#IntelligentAgent 2 #AWS 1 #Docker 1 #Flask 1

Fig. 3.2.: Skills Section


3D DRAWING

Last Edit: Feb 14, 2020 at 3:10pm

Graphics strengthens one's imagination.

In this course project, I draw three separate parts and assembled them to gather. These parts included a glass, the shape of the orange juice, and the saucer. The idea of the project came from the fact that Iranians traditionally drink tea in a glass and a saucer — unfortunately, the version of #Solidworks. I used didn't provide a tea color; therefore, I replaced it with orange juice.

Guess which one is the original picture! :-?

Skills Demonstrated:

#SolidWorks 56 #Designing 40

Fig. 3.3.: Showcase Example along with Demonstrated Skills

One final section is the Social Engagement Section, where the closest people in your network are illustrated, along with their ranking. This closeness depends on your engagement with the users. For example, if you reflect on a particular user's posts or rate them a lot, your engagement score with that user increases. As a result, that user's ranking will improve on this section. Figure 3.4 shows this section with top-six closest users.

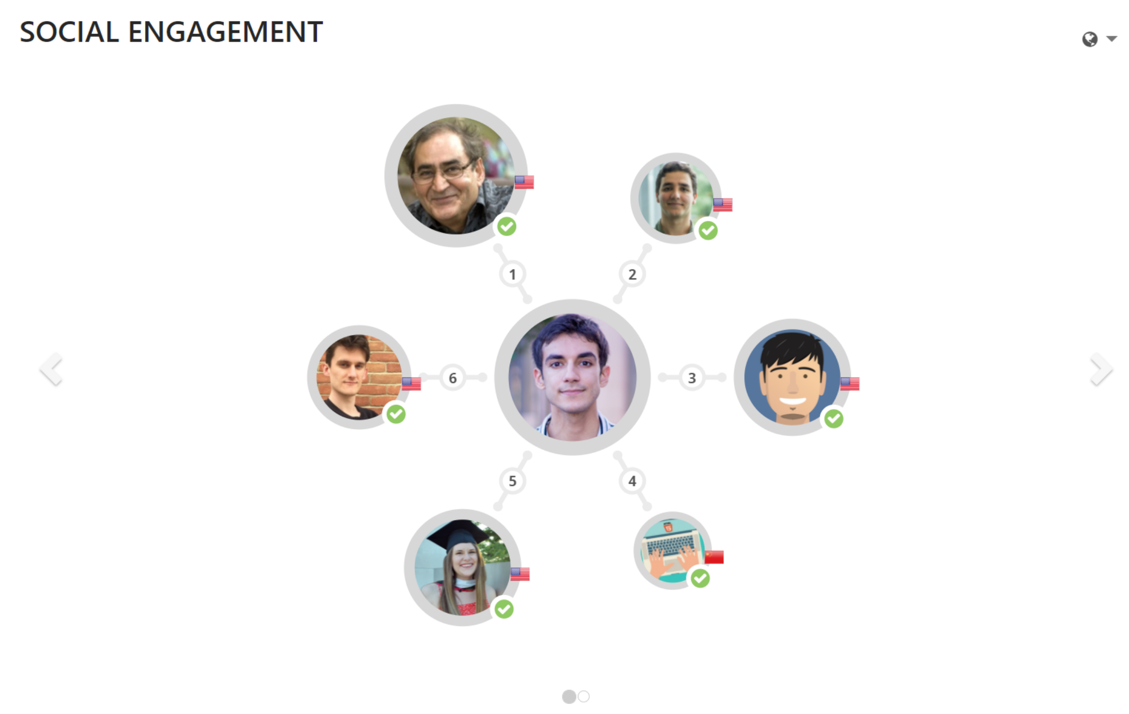


Fig. 3.4.: Social Engagement Section

3.3 Social Network

CN is also a social network where people with similar interests or goals are gathered in a network, course, or community. In each of these groups, members can share their thoughts, documents, and communicate by creating posts, polls, or events. Others can reflect and rate the created posts, polls, and events. Every CN member can create any of these groups which can be either private or public, except for communities. The main differences among them are:

- Networks usually are associated with a licensed institution.
- A course belongs to an instructor or a lecturer that can be a member of a network or not.
- A community consists of people that relate common interests or goals. All the communities on CN are publicly available for all the users to join them.

All of these actions take place on the Home Feed where the social network aspect of CN along with the LMS aspect is combined. That is, all the information about the mentioned groups is displayed.

Typically, users join networks and courses because of their institute or the course they have taken and they cannot join any desired network or course; unless they are public. On the other hand, as stated above, communities are publically available and anyone with a CN account can join them. Users' communities are listed on top of the Home Feed (Figure 3.5). If a user clicks on any of them, the content of the Home Feed is filtered and only the related material to the selected community is shown. On the button right of Figure 3.5, there are two buttons. Users can remove or sort their communities with the "Edit" option and they can add more community by using the "Join Communities" button.

Throughout this project, the phrase "tag" is used for referring to a community. This tag can be a single word or a long-phrase and they can be categorized into the following categories followed by an example in the parenthesis:

- Skills: This kind of community discusses a specific skill and whoever is interested in learning that skill can join them (e.g. Python). These tags are the same as the skills on the ePortfolio.

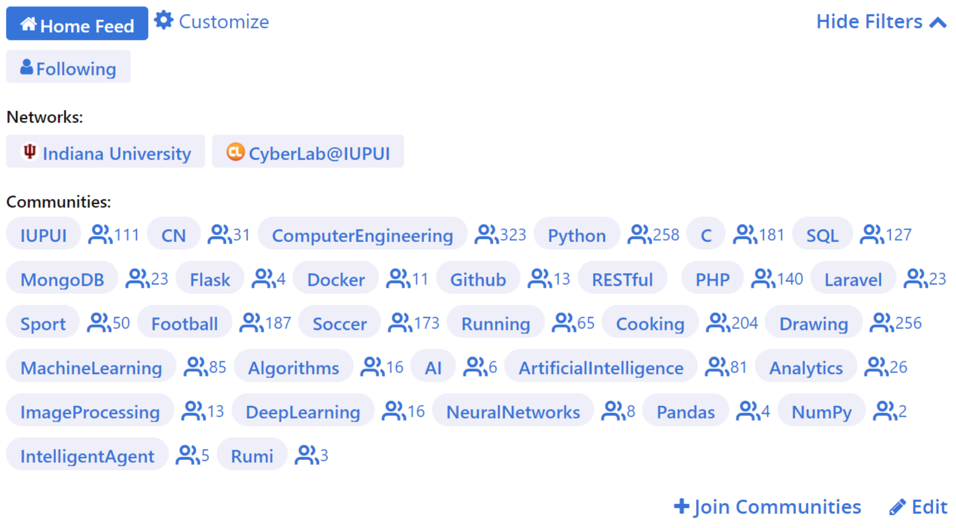


Fig. 3.5.: Community Section

- Majors: Field of studies are another category which sometimes they are accepted as a skill. An example of that is computer science which is both a skill and a degree.
- Companies: If a community belongs to a company, job-seeking users can join their community and get more familiarized with that company. (e.g. CourseNetworking)
- Events: Name of any event such as conferences and meetings is a great choice of community where the members or audiences can join and interact (e.g. IU Online).
- Locations: Some communities are locations of an event or they are basically the name of a city, region or country (e.g. US).
- Sports: Sports are another community to connect members of a system. (e.g. Soccer)
- Others: The fact that one can make a new community with any name in their mind, makes this feature of CN very interesting.

To conclude this chapter, CN is an LMS and a social network that supplies an ePortfolio. In essence, CN is a combination of LinkedIn, Facebook, and Canvas in terms of having an ePortfolio, being a social network, and an LMS, respectively. This project focuses on the skill and about parts of the ePortfolio and the community section on the Home Feed.

4. RUMI

4.1 Intelligent Agent

In the previous chapter, the crucial role of Learning Management Systems (LMSs) in today's environment and differences of CourseNetworking (CN) with a conventional LMS was discussed. In this chapter, a new tool is explained that can help an LMS become more dynamic.

Generally, LMSs are very static, which means the functionalities of the system are equal for every user. For example, when a file is added to a course by the instructor, the system sends a message to all the students who did not disable their notification. But this message does not consider the status, mood, and availability of a user. Moreover, these systems do not have any persona and the interaction between a user and an LMS does not mimic a human communication. Having non personalized reactions and pre-defined instructions for every command will not build the desired trust, mentioned in Section 2.2.

In order to have a smart LMS in the sense that the functionalities differ for different users, CN has come up with the idea of utilizing artificial intelligence (AI).

4.2 How can AI help?

AI can help to add the properties mentioned in Section 2.2 to a learning system. This aid can be done in various ways such as advising, networking, assisting, and entertaining. AI can advise the students by notifying their due dates in a course, encouraging them to learn the materials, and studying the grade trends to find the outliers and alert the user in those conditions.

Also, AI can network the users of a system by recommending friends to follow or send messages. It can even connect users who are looking for and offering tutoring in a specific field. Assisting can be done by helping to build an ePortfolio. As an example, there are many CN accounts that have not completed their ePortfolio. AI can assist by recommending jobs to job seekers. Finally, holding some AI-based competitions can make these systems more amusing.

4.3 Rumi

The AI project of CN in collaboration with CyberLab is about developing and conceptualizing an intelligent agent called Rumi, who can make an LMS more dynamic and tailored toward users' needs. The name is taken from the famous Iranian poet, Jalāl ad-Dīn Muhammad Rūmī, who is known for his wisdom.

In CN, Rumi plays the role of a digital mentor, a personal assistant, and an advisor [24]. Rumi has a human-like persona and is a male in his mid 30's. He expresses his emotion through facial expressions. That is, he can be either neutral, sad, or happy based on the content of the service he provides [24]. Figure 4.1 shows different faces of Rumi.

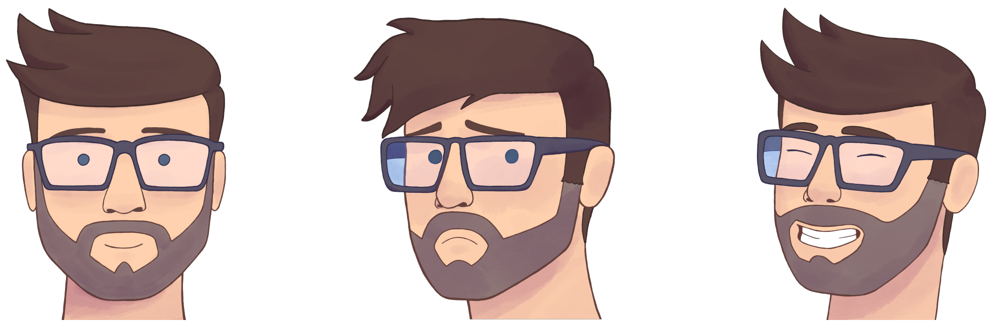


Fig. 4.1.: Rumi's Different Facial Expressions. From left to right: Neutral, Sad, and Happy

In general, Rumi communicates with users by offering some services within a unique post. This post appears on top of the CN's website. Examples of some designs are included in Section 5.7. These services include:

- Notifications
- Community Recommendations
- Job Recommendations
- Quote Competition
- Tips

In Notifications, Rumi tells the users who have checked their ePortfolio or gave them a new recommendation. He also notifies them if they got a new endorsement for any of their showcases, expertise, or recommendations. Another typical notification is about people who start following others.

The primary purpose of this work is about Rumi's Community Recommendations. In which, Rumi suggests new communities to a user based on their CN profile and connections. A comprehensive explanation of this service, including its designs, is given in Chapter 5.

In the Job Recommendations service, Rumi gathers users' information and directs them to job engines. These engines include, Indeed [25], Chegg Internships [26], and HigherEdJobs [27].

Although Rumi seems serious at first glance, he also amuses users by holding competitions. One of which is the Quote Competition [28], where it promotes the gamification aspect of Rumi. In this competition, Rumi collects users' quote, shown in Figure 3.1 [28]. Then based on an ELO system, he recommends two quotes at a time to each user and asks them to select the better quote [28]. Finally, with the results obtained, the top quotes receive a badge from CN [28].

In Tip service, Rumi briefly describes some information. This information includes informing users of CN's features, recommending resources, or sharing learning and teaching guidance.

4.4 Implementation Process

As mentioned in Section 2.2, the architecture and integration of an intelligent agent required developing the front-end and writing the back-end in PHP with the use of Laravel framework. Implementing Rumi in this way has several limitations. First of all, CyberLab graduates are usually not experienced front-end developers, and writing the front-end code will distract them from their primary role as machine learning researchers. Second, popular machine learning libraries and packages are written in Python. Third, connecting the scripts written in other programming languages than PHP to Laravel would require a layer of application programming interface (API). Having many API layers results in less efficient integration with CN and other learning systems.

The implementation process of Rumi was redesigned in July 2019, so the process becomes faster and resolves the limitations mentioned above. As of July 2019, the CyberLab team has decided to use Python as the programming language for back-end development and provide the API for integration with other platforms by utilizing one of the most popular Python web application frameworks, Flask. Rumi uses a MongoDB database due to the same reasons discussed in [22] and is run on Amazon Web Services (AWS).

The software has been dockerized with Docker so that future CyberLab graduates can easily catch up with the process and add more services to Rumi. Having an API-based architecture that only contains the services' content will not involve any front-end development and make the transition to other programming languages and frameworks more manageable.

The front-end is being implemented by CN's production team in overseas collaboration with the CyberLab team. Therefore, more time is devoted to the machine learning aspect of the project and providing the APIs. All of these changes have taken into account that Rumi functions as software as a service (SAAS), one of the Rumi project goals.

4.5 Inference Engine

In general, Rumi interacts with users with one of its services. The challenges for Rumi are when to interact and which service to choose. To tackle these challenges, an "Inference Engine" is announced. Similar to the priority engine that is described in [22], Rumi's inference engine picks the most important service based on the feedback received from each CN member. A cycle is defined for when to show the chosen service to the user. This cycle is set at 12 hours, which means the time between each time Rumi appears should be at least 12 hours. Whenever a user logs in, if this time has passed since the last time Rumi displayed a service, the inference engine is called, and the next service is shown. There is an exception to this scenario, which is when a notification exists. In that case, no matter when was the last time that Rumi appeared, he will deliver that message to the user.

This engine also collects the feedback for each service and does some calculations that [22] refers to as the reasoning engine. Any feedback is related to either a service or, in general, Rumi's settings. When the inference engine receives feedback from the user, it sends it to the right service, or if it is related to the settings options, it will apply the necessary changes. The feedback received by a particular service is processed, and the result is sent back to the inference engine to help with the inference of the next service. In the end, the feedback is stored in the database. Figure 4.2 illustrates this process.

For every CN account, the inference engine computes a Service Score for each of Rumi's services except the Notifications service. Every time the inference engine is

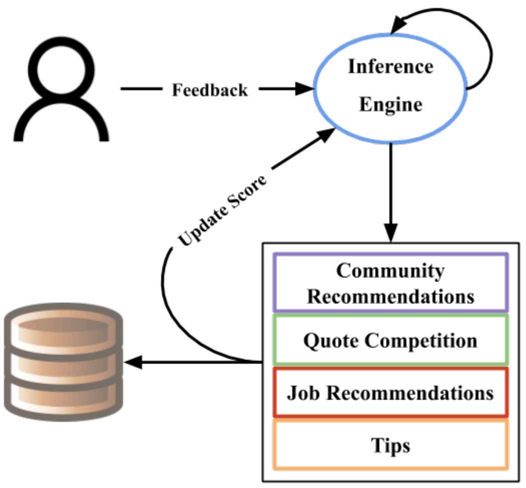


Fig. 4.2.: Feedback Structure

called, the service with the highest Service Score is inferred as the most valuable service. Then the content of the selected service is displayed to the user. The algorithm behind the scoring mechanism considers three main factors:

- Repetition: The number of times that a specific service is shown back-to-back. The algorithm has to rotate the services in such a way that all the services have a chance to be chosen, and it does not go into a loop that continually picks exactly one service.
- Interaction: The number of interactions a user makes with a particular service. The interactions here refers to the requests sent to the server either by selecting buttons or scrolling the pages. Some of the services require many clicks, while others can have as low as one click. Hence, the algorithm has to come up with a scaling factor to balance these interactions.
- Flag: Users, CN admins, or institution admins can disable a service. If the flag is off for a service, the algorithm should not select that service until the status of the Flag changes.

These factors are employed to formulate the Service Score. To do so, a score is assigned to each factor. The Service Score is a real number and is called *Score* in our formulation. It is assumed that *Score* has a non-negative value, so its minimum is zero.

Also, Repetition, Interaction, and Flag scores are referred to as *repetitionScore*, *interactScore*, and $flagScore$ respectively. The $flagScore$ is a Boolean variable. Its default value is set to one, and whenever the service is turned off, it will become zero. Because *Score* is always greater than zero, *repetitionScore* and *interactScore* are non-negative too. The Service Score is calculated using Equation 4.1 only for the services which their $flagScore$ is non-zero. Then based on Algorithm 1 the next service is chosen.

$$Score = interactScore * repetitionScore \tag{4.1}$$

Algorithm 1 Inference Engine Algorithm

```
Input: SERVICES                         ▷ The SERVICES includes all types of scores
Output: $chosenService$
 1: $Max \leftarrow 0$
 2: $Score \leftarrow 0$
 3: $chosenService \leftarrow \emptyset$
 4: for each $service$ of SERVICES do
 5:     if $service.flagScore$ is not zero then
 6:         $Score \leftarrow service.repetitionScore \times service.interactScore$
 7:         if $Score$ greater than $Max$ then
 8:             $Max \leftarrow Score$
 9:             $chosenService \leftarrow service$
10:         end if
11:     end if
12: end for
```

As shown in Figure 4.2, although every feedback is stored, the inference engine relies on certain ones to measure *repetitionScore* and *interactScore*. First, a fixed-length history list is defined to hold the number of repetitions and interactions for every service. The length of this list is called l. In other words, every time a service is inferred, both service's name and the number of interactions that the user had with it are stored.

The *repetitionScore* for service i is measured using the last l inferences. If that service is chosen in the jth inference, r_{ij} will be one otherwise zero. This score aims to provide a chance for services that have not been shown in a while and prevent the algorithm from reselecting a particular server over and over. For the former goal, a linear function or an exponential one is suitable. The latter end can be achieved by assigning a score of zero to the service shown in the entire history list.

In practice, the exponential function below achieved a better result when utilized with the formula of *interactScore*. The *repetitionScore* of service i is computed using Equation 4.2.

$$repetitionScore(i) = 1 - \frac{\sum_{j=1}^{l} 2^{r_{ij}}}{2^l - 1} \tag{4.2}$$

The *interactScore* goal is to increase the chance of being inferred for the services with a high number of interactions. Here, the first few interactions are more precious than those received afterward. More so, the closer the number of interactions gets to the maximum limit, the slower this score should grow. This phenomenon is observed in exponential functions where at first, their value increases fast, but after a while, it is saturated.

The formula applied to calculate the *interactScore* is shown in Equation 4.3, where a and b are parameters that along with l have to be tuned. Also, s_{ij} is the total number of interactions for the ith service in the jth inference. If a service is not picked in an inference, this value will become zero.

As mentioned, s_{ij} will vary for every service, and the sigma in Equation 4.3 intensifies these differences. Thus, there should be a way to compensate. A solution is to multiply a constant to the total score of each service. This constant controls the service's total score to stay in a certain range. The c_i in Equation 4.3 is a constant associated with service i.

$$interactScore(i) = 1 + \frac{\exp\left(1 + c_i \sum_{j=1}^{l} s_{ij}\right)}{a \exp\left(b\right)} \tag{4.3}$$

The next chapter discusses the algorithm behind the Community Recommendations and showcases user-friendly designs.

5. COMMUNITY RECOMMENDATION

5.1 Introduction

As discussed in Section 4.3, Rumi provides many services. One of which is Community Recommendation. Referring to Section 3.3, every user has a section related to their communities on the Home Feed. They can add or modify their joined communities with editing and joining options. Typically, CourseNetworking (CN) members join these communities to communicate, socialize, or learn new things, especially if the community is related to a specific skill. By looking at CN's data, it is noticed that many users haven't joined any communities, and most of the communities have a few members. Rumi, as an intelligent agent, is here to connect people and make the system more productive and engaged. The more engaging users become, the more fruitful the social network becomes. Consequently, recommending communities can be beneficial to fulfill these goals.

Not any recommendation can encourage members to join a community. These recommendations have to be related to their taste, personality, profession, skill set, and surroundings. Also, the way recommendations are shown can attract them to add them. If a list of many communities is shown at once, it can distract the user's attention. Moreover, a totally unrelated recommendation might lessen the acceptance rate.

Rumi has to consider all of the conditions mentioned above. Hence, data preprocessing is vital to prepare the recommendation list. After the data is cleaned, then machine learning techniques become handy to find the most suitable communities for each member. The advantage of recommending communities is that there is no need to check the tag's semantic meaning as long as it does not have a typo. Whereas when it comes to recommending skills, it is essential that the label be a skill. For example, if a recommender system suggests “IUPUI” to a user as a skill recommendation, it has low accuracy and negatively influences the user. Any destructive impulse received by a user from Rumi will shake the trust between the agent and that user. Thus, Rumi must employ the most accurate recommender engine.

This chapter’s main objective is to devise a recommender system for Community Recommendation by utilizing three methods and then comparing their accuracy. These methods are a modified version of Collaborative Filtering (CF), Continuous Bag of Words (CBOW), and transfer learning using Bidirectional Encoder Representations from Transformers (BERT).

5.2 Data Preprocessing

This project uses sample data from CN’s testing database. It consists of users’ ids, their Fields of Study, user’s social engagement, and, most importantly, their list of communities. The ids were mapped to natural numbers to keep users’ identities secret. The Field of Study and social engagement info is taken from the About and Social Engagement Sections on the ePortfolio. This sample data extracted from the CN is stored in CSV format.

The data is passed through a preprocessing pipeline before being used in the recommender systems. In this process, users with zero numbers of communities and no Fields of Study were initially removed because there is insufficient information regarding them. Next, all the Non-English communities and Fields of Study were removed by comparing their characters to English characters.

For the remaining communities and Fields of Study, their characters were converted to lowercase to check for typos and duplication. A heuristic threshold was chosen to remove the communities that their members’ number is less than the threshold. This heuristic value not only removes the tags that have typos but also discards meaningless communities or those that belong to test accounts.

The total number of distinct communities in the sample data is more than 14,000. After passing it through the preprocessing pipeline explained above, this number drops to less than 1,000 tags. The same scenario holds for different majors in the data. Before the preprocessing, the total number of majors is above 9,000, but after crossing the pipeline, it reduces to around 3,500.

5.3 Collaborative Filtering

The first approach is a modified version of item-to-item CF [8] called Neighboring Collaborative Filtering (NCF). The name is derived from the general idea, which selects popular tags among the user's neighboring members as community recommendations. The neighboring members of a target user are people who are somehow related to that user. There are many features on CN that can cause two users to be related. Some of which are as follow:

- Skill: When both users have at least one skill in common on their ePortfolio. These skills are shown in Figure 3.2.
- Community: Same as the above feature, instead they are members of at least one community in the system. Also, these communities are retrieved from the Community Section, Figure 3.5.
- Field of Study: Similar to the last two features, in this feature, users are related when they have specified the same Field of Study in their About section, Figure 3.1.
- Social Engagement: As shown in Figure 3.4, a user is connected to people who they have engaged more.
- Colleague: When two users have enrolled or are enrolling in the same course in CN, they are related. This relationship also includes the relation between an instructor and a student.

- Following and Followers: If any of them follows the other's account, both are related.
- Endorsement: When one of the users has endorsed any section of the other user's ePortfolio, this makes a relationship between the two.
- Recommendation: When one has written a recommendation for the other user, a connection has occurred.
- Post: If any of the two has rated or reflected on the other user's posts, the two are linked.
- Organization: When two users are former or current members of an organization, there is a connection between them.
- Nationality: When both have the same nationality, they are linked.
- Location: When two users have declared the same state or city on their About section, they are related.

By studying the skill relation, it is seen that in most of the CN accounts, all of a user's skills are also shown on their community list. One of the reasons for this observation is that whenever a user adds a skill to their ePortfolio, the system automatically adds that tag to their community list. Thus, the skill list is a subset of the community list unless someone manually adjusts these two sections on their account.

In the sample data, there are many inactive members. Inactive members are people who do not interact with the system regularly. As a result, each member has slightly over one follower on average because of inactivity. Also, after cleaning the data, there is not much improvement in this average. So, this project ignores the Following and Followers feature.

Endorsing some parts of the ePortfolio and writing a recommendation are newly added features to CN. More so, writing a recommendation takes some time, and users resist doing so. For these reasons, the sample data does not have sufficient data related to these two features.

The rest of the features are too general to be accepted as a reasonable factor for finding members with common community interest as the target user. Hence, out of all of the features mentioned above and many more, only the following were chosen to find the neighboring members.

- Community
- Field of Study
- Social Engagement

In other words, users who have a common Field of Study or community or have engaged with the target user are considered as the neighboring.

The next subsection describes NCF in detail, but it is worth getting an overview of the method prior to that. Figure 5.1 illustrates the idea of how the NCF algorithm works.

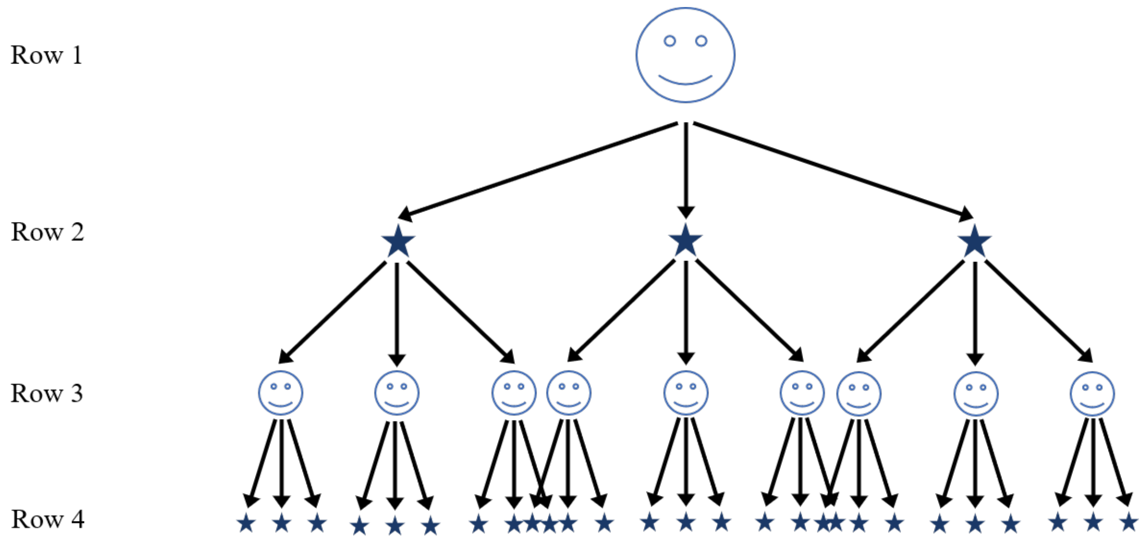


Fig. 5.1.: Intuition Behind the Modified CF Algorithm

The first row shows the target user. All the user's data that can be used to find related members are exhibited in the second row. These features are the three mentioned above. Most importantly, the neighboring members come in the third row. Eventually, the most frequent communities among these members are selected as community recommendations for the target user unless the target user has already joined them.

5.3.1 Algorithm

The basic algorithm using nested for loops is shown in Algorithm 3. This algorithm uses a function shown in Algorithm 2, where the function inputs a user and a specific feature. Then it returns the related users to the input user based on that feature. For example, assume a user is majoring in Computer Science and Computer Engineering. If this user and the feature of major is passed to $NEIGHBORING$, the inner loop searches for members who are majoring in either Computer Science or Computer Engineering. The result is the users who have specified one of the mentioned majors in their Field of Study.

Algorithm 2 Neighboring members

```
 1: function NEIGHBORING($user$, $feature$, USERS)        ▷ Fixed $user$ and $feature$
 2:     $Neighboring \leftarrow \emptyset$
 3:     for each $item$ in $user.feature$ do
 4:         for each $member$ of USERS do
 5:             if $item$ exists in $member.feature$ then
 6:                 add $member$ to $Neighboring$        ▷ Duplicate members can exist
 7:             end if
 8:         end for
 9:     end for
10:     return $Neighboring$
11: end function
```

Assuming that there exist n users and each user can have at most k instances of a feature, the function runs in $O(nk^2)$ in the worst case. One can look at k as the number of majors, communities, or relationships that a user can have.

In Section 5.3.3, a creative way is explained to reduce this running time to $O(k)$ by storing data in the database.

Algorithm 3 NCF Algorithm

Input: USERS, FEATURES

Output: recommendation list for each user

```
1:  for each $user$ of USERS do
2:      $RecommendationList(user) \leftarrow \emptyset$
3:      $List \leftarrow \emptyset$                                  ▷ Duplicate values can exist
4:      for each $feature$ of FEATURES do                         ▷ Fix one of the features
5:          for each $member$ of NEIGHBOROFG($user$, $feature$, USERS) do
6:              for each $community$ of $member.community$ do
7:                  if $community$ don't exist in $user.community$ then
8:                      add $community$ to $List(user)$
9:                  end if
10:             end for
11:         end for
12:     end for
13:     for each $unique\_community$ in $List$ do
14:         count the number of repetitions of $unique\_community$ in the $List$
15:         add $(community, count)$ to $RecommendationList(user)$
16:     end for
17:     sort the $RecommendationList(user)$ in descending order based on $count$
18: end for
```

The Algorithm 3 can be divided into three parts to calculate its runtime. First is the procedure to find the *List* of communities. Then, the process of counting the repetition of this *List*. Finally, sorting the recommendation list.

The first part is run in $O(n^2k^4mf)$, where f and m are the number of features and maximum neighboring members for a specific feature, respectively. The second and third parts depend on the number of unique communities, c. The former is run in $O(nh)$ if counting sort is used, where h is equal to $\max\{c, mfk\}$. The latter will have a runtime of $O(nc\lg c)$ with insertion sort. In total, because n is much greater than c, the runtime in the worst case will be $O(n^2k^4mf)$. Again, if some repetitive calculation is stored in the database, this runtime will decrease a lot.

Another way to reach the same result is to employ matrix multiplication. This approach, Algorithm 4, is much faster than Algorithm 3 because Python performs matrix multiplication smarter than nested for loops. Note that this algorithm can be utilized for other recommendations, such as major, course, and book recommendations. Here it is assumed that the SUBJECT is equal to "community".

As Algorithm 4 shows, the first matrix U has all users on both of its rows and columns. At first, this matrix is initialized with zero. Every time a feature matrix is created, the multiplication of this feature matrix to its transpose is added to U. The constructed matrix, after multiplication, holds the number of common items between two users. For instance, if the feature is equal to major, the multiplication of the major matrix to its transpose will create a matrix that shows how many common major two users have.

Each feature matrix has n rows and m columns, where n and m are the number of users and unique items in that feature, respectively. In the end, the final U is multiplied to the community feature matrix to create R. The community feature matrix shows the relation between users and all the communities.

It is important not to recommend the already joined communities to a user. So, R should be multiplied one more time to the community feature matrix, and the result must be subtracted from R to remove the joined communities.

Algorithm 4 Matrix Approach for NCF Algorithm

```
Input: USERS, FEATURES, SUBJECT
Output: $R$                                        ▷ recommendation matrix
 1: $n \leftarrow length(\text{USERS})$
 2: $i \leftarrow 0$
 3: Create $U[n][n] \leftarrow 0$                      ▷ creating user relation matrix
 4: for each $feature$ of FEATURES do                  ▷ Fix one of the features
 5:     $i \leftarrow i + 1$
 6:     $m \leftarrow length(feature)$
 7:     Create $F_i[n][m]$                     ▷ creating feature matrix for each feature
 8:     for each $user$ of USERS do
 9:         for each $item$ of FEATURES do
10:             if $item$ exists in $user.feature$ then
11:                 $F_i[user][item] \leftarrow 1$
12:             else
13:                 $F_i[user][item] \leftarrow 0$
14:             end if
15:         end for
16:     end for
17:     $U \leftarrow U + F_i \cdot F_i^T$           ▷ adding weights to the relation between users
18:     if $feature$ equal to SUBJECT then                 ▷ Is true only once
19:         $index \leftarrow i$                           ▷ storing the SUBJECT matrix
20:         Create $R[n][m]$                         ▷ creating recommendation matrix
21:     end if
22: end for
23: $R \leftarrow U \cdot F_{index}$
24: $R \leftarrow R - R \cdot F_{index}$       ▷ removing users' SUBJECTs from recommendations
```

5.3.2 Feedback

The NCF algorithm can be further improved by utilizing a feedback structure. This part explains how this feedback is received and embedded in the NCF algorithm for both approaches.

As mentioned before, Rumi learns from the user's feedback and improves his services' quality. In Community Recommendation, if a user does not select a community after they saw it, it can be inferred that they did not want to join it. The unchosen community is referred to as a removed community.

By applying this feedback to Algorithm 3, it changes to Algorithm 5. The only difference is in lines 17-23, where the feedback is applied. For every removed community, the communities of its members are discarded from the *List*. In other words, referring to Figure 5.1, let's call every item in the second row and all of its descendants a Branch. Then lines 17-23 are saying which Branches to remove from the figure.

It is assumed that line 20 is run in $O(1)$ with a hashing mechanism. Similar to the NEIGHBORING function of Algorithm 2, lines 18-22 will take $O(nk^2)$. Because a user can send feedback to all c communities, the outer loop is run c times. After all, the new lines run in $O(cnk^2)$, which is not comparable with $O(n^2k^4mf)$ and is much smaller.

In the matrix approach, the story is slightly different. When the feedback is included, Algorithm 4 alters to Algorithm 6. The new algorithm is only storing the non-zero values in lines 8-12. The reason for this will be discussed in Section 5.3.3. When creating the U matrix, if the feature matrix is about the relationship between communities and the users, then the feedback is applied to that multiplication to reduce the time and make the algorithm more efficient. That is, first, the feedback matrix, D, is created from the removed communities. Then D is subtracted from the community feature matrix. The resulting matrix is multiplied to its transpose and then added to U. This matrix can have negative entities, while the feature matrices do not have any.

Algorithm 5 NCF Algorithm with Feedback

Input: USERS, FEATURES

Output: recommendation list for each user

```
1: for each $user$ of USERS do
2:     $RecommendationList(user) \leftarrow \emptyset$
3:     $List \leftarrow \emptyset$                                  ▷ Duplicate values can exist
4:     for each $feature$ of FEATURES do                 ▷ Fix one of the features
5:         for each $member$ of NEIGHBOROFG($user$, $feature$, USERS) do
6:             for each $community$ of $member.community$ do
7:                 if $community$ don't exist in $user.community$ then
8:                     add $community$ to $List(user)$
9:                 end if
10:            end for
11:        end for
12:    end for
13:    for each $unique\_community$ in $List$ do
14:        count the number of repetitions of $unique\_community$ in the $List$
15:        add $(community, count)$ to $RecommendationList(user)$
16:    end for
17:    for each $tag$ in $user.feedback$ do
18:        for each $member$ that $tag$ exists in $member.community$ do
19:            for each $community$ in $member.community$ do
20:                $(community, count) \leftarrow (community, count - 1)$
21:            end for
22:        end for
23:    end for
24:    sort the $RecommendationList(user)$ in descending order based on $count$
25: end for
```

Algorithm 6 Matrix Approach for NCF Algorithm with Feedback

```
Input: USERS, FEATURES, SUBJECT
Output: $R$                                          ▷ recommendation matrix
 1: $n \leftarrow length(\text{USERS})$
 2: $i \leftarrow 0$
 3: Create $U[n][n] \leftarrow 0$                      ▷ creating user relation matrix
 4: for each $feature$ of FEATURES do                  ▷ Fix one of the features
 5:     $i \leftarrow i + 1$
 6:     $m \leftarrow length(feature)$
 7:     Create $F_i[n][m]$                              ▷ creating feature matrix for each feature
 8:     for each $user$ of USERS do
 9:         for each $item$ in $user.feature$ do
10:             $F_i[user][item] \leftarrow 1$
11:         end for
12:     end for
13:     if $feature$ equal to SUBJECT then              ▷ Is true only once
14:         $index \leftarrow i$                        ▷ storing the SUBJECT matrix
15:         Create $R[n][m]$                            ▷ creating recommendation matrix
16:         Create $D[n][m]$                            ▷ creating feedback matrix
17:         for each $user$ of USERS do
18:             for each $tag$ in $user.feedback$ do
19:                 $D[user][tag] \leftarrow 1$
20:             end for
21:         end for
22:         $U \leftarrow U + (F_i - D) \cdot (F_i - D)^T$    ▷ adding feedback weights to the user relation matrix
23:     else
24:         $U \leftarrow U + F_i \cdot F_i^T$          ▷ adding weights to the relation between users
```

Algorithm 6 Matrix Approach for NCF Algorithm with Feedback (continued)

```
25:     end if
26: end for
27: $R \leftarrow U \cdot F_{index}$
28: $R \leftarrow R - R \cdot F_{index}$        ▷ removing users' SUBJECTs from recommendations
```

By ignoring the time it takes to calculate the matrix multiplication, Algorithm 6 runs in $O(fnk)$, where f is the number of features, n is the number of users, and k is the maximum number of feature instances that a user can have.

Note that it is also assumed the zero matrices are instantly created, and no time is required to store that many zero entities. In the next section, the way to store and use the matrices are explained.

5.3.3 Implementation

The two approaches for the NCF algorithm that consider feedback, Algorithm 5 and Algorithm 6, are implemented in CN. Each of which had their challenges. This part discusses the solutions to overcome these challenges.

Both algorithms generate recommendations for all the users, and they both sort the final recommendation list. These two actions are not necessary to take place in the same way stated in the system. First of all, a new recommendation list is required whenever a user interacts with the system and changes something on their profile. Therefore, the algorithm needs not run for all users, which reduces the runtime by a factor of n. Second, after all, the algorithm tries to find the most useful communities for the user. Thus, sorting the recommendation list is useless, and only selecting the top-ranked communities is sufficient.

By applying these two adjustments, both approaches will run much faster. Here, the time to connect to the database or retrieve data is neglected because both algorithms perform the same operations. The generating process is taken place offline after a specific period, one hour, and the system creates new recommendations for the users whose trigger is set. In contrast, some parts of the feedback mechanism, such as removing the already joined communities from the recommendation list, are applied instantly.

Because CN uses MongoDB, then the terms collections, key, and type are used here. A Collection is equivalent to tables in relational databases [29]. Records are stored as a BSON document in MongoDB collections [30]. BSON is a binary representation of JSON documents, and they have different types [31].

Nested For Loop Approach

The running time of Algorithm 2 and Algorithm 5 are $O(nk^2)$ and $O(n^2k^4mf)$, respectively. If the data needed for the $NEIGHBORING$ function is stored in memory, by assuming that the time for retrieving data is $O(1)$, then both runtimes will drop by a factor of nk. In other words, the goal is to make the inner loop of Algorithm 2 run in $O(1)$.

A collection associated with each feature is required to achieve this end. This collection should contain all the users who have a specific instance of that feature. The time needed to create each of these collections relies on the total number of instances, d, for every feature. Then with one scan over the user collection, all the users having that instance can be found. Thus, the total time will be $O(dnk)$. Certainly, this runtime could be lessened, but because it is only run once a week or month, it is left as is. Tables 5.1 and 5.2 show two such collections.

For example, the Community User IDs Collection stores all the users who are members of a specific community. This collection contains the name of a community along with the id of users in that community. This collection can help to decrease the running time of lines 18-22 in Algorithm 5 from $O(nk^2)$ to $O(k)$ because it is not required anymore to search for members of a community.

Table 5.1.: Community User IDs Collection

Community_UserID

Field	**Type**
_id	ObjectId
Community	String
User IDs	Array
Count	Int32

Table 5.2.: Major User IDs Collection

Major_UserID

Field	**Type**
_id	ObjectId
Major	String
User IDs	Array
Count	Int32

To further reduce the running times, both of the above adjustments are taken into considerations. A trigger is added to each user's document. Whenever a user adds or discards one of the features attached to the algorithm, this trigger will be set. So, except the first time the algorithm is run, the outer loop in Algorithm 5 would only consider users who their trigger is set. As a result, this could boost the speed by ten times.

Moreover, only 60 of the top recommendations are stored in the Community Recommendations collection for each user. This collection is shown in Table 5.3. Instead of sorting all the tags in *RecommendationList*, the top 60 ones are found and saved for future reference.

Table 5.3.: Community Recommendations Collection

Community Recommendations

Field	**Type**
_id	ObjectId
User ID	String
Communities	Array

Matrix Approach

One challenge in this approach is the memory requirement. Each feature matrix in Algorithm 6 can have more than nd entities, where n and k are the number of users in the system and number of instances of a feature. After multiplying this matrix to its transpose, the number of entities reaches n^2. For a system with a million users, storing this amount of entities in the random access memory (RAM) is costly.

The bright side is that these matrices are sparse and have many zero elements. With the notation used before, every row of them can have at most k ones. Even when they are multiplied together, the result is still a sparse matrix. Thus, matrix U will not have many non-zero entities and is a sparse matrix.

A solution to overcome this challenge is to utilize libraries that efficiently calculate sparse matrices multiplication. Here, the SciPy's Compressed Sparse Row matrix (csr_matrix) [32].

Although both modifications to decrease the runtime hold here too, this approach has a disadvantage. That is, the result matrix after each multiplication should be stored on RAM. To handle this issue and save memory, one can immediately delete that matrix after it is added to U. In the end, the recommendations are stored in the Community Recommendations collection.

5.4 Continuous Bag-of-Words (CBOW)

Natural language processing (NLP) is one of the subfields of artificial intelligence. Recommender system engines have different components in which processing natural language efficiently plays an important role, such as language understanding and language generation [33].

In recent years, NLP algorithms have been used massively in designing and training recommender systems. Recommender systems that feed in textual data as inputs require understanding items and user profiles and processing their information. This system can be formulated using NLP word2vec algorithms, where each word is represented as a continuous vector. At the next step, the relation between the items is required to be under consideration to rank them by order of the most relevant to the user's intent [34].

Two of the NLP models, Continuous Bag-of-Words (CBOW) and Continuous Skip-gram, are proposed in [12]. The former tries to predict the target word based on the context, while the latter is the way around. Given the target word, it predicts all the surrounding words [12]. Figure 5.2 illustrates these two architectures with their three different layers; input, projection, and output.

To exploit an NLP model for community recommendations, one can look at a user's communities as a sentence. For example, assume Emad Azad in CN is a member of Computer Science, Python, Machine Learning, Artificial Intelligence, and TensorFlow communities. This information from the user can be converted to a sentence like below:

> "Emad Azad is a member of Computer Science, Python, Machine Learning, Artificial Intelligence, and TensorFlow communities."

After this conversion, both CBOW and Skip-gram model can recommend a new community to Emad. The difference will be in how the models are trained and what will be the input and output.

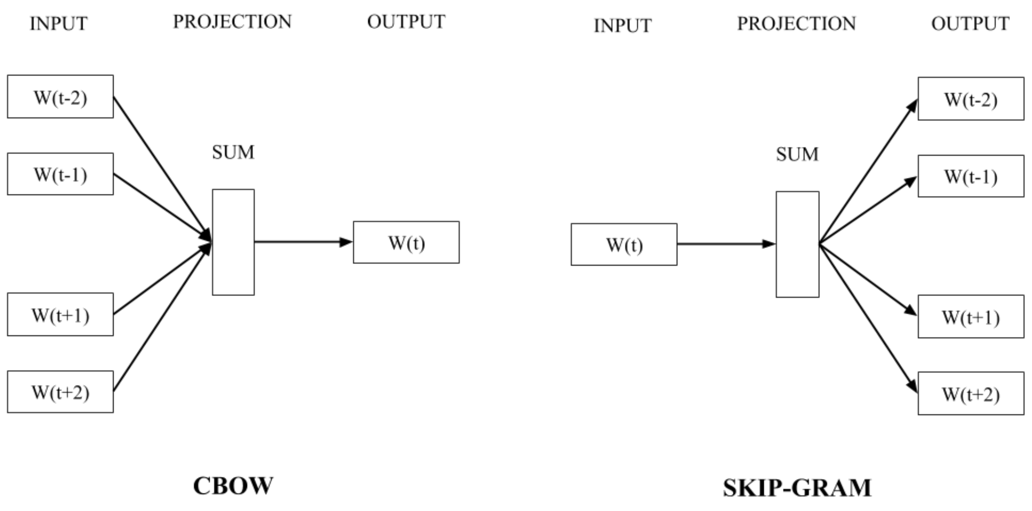


Fig. 5.2.: CBOW and Skip-gram Architectures

In general, as shown in Figure 5.2, the CBOW model collects more information and outputs the most probable word. Moreover, the term bag-of-words refers to the fact that the words' order in the input layer does not influence the projection layer [12]. In contrast, Skip-gram requires less information, only one word, and outputs several options. It is found that increasing the length of the output layer improves the quality of the prediction. However, because this length is correlated to the model's complexity, it will be computationally more expensive [12].

When it comes to the community recommendation problem, the goal is to find the most suitable community for the user. Also, due to the conversion explained above, the ordering of the communities does not matter. That is, all three sentences below have the same meaning:

"Emad Azad is a member of Machine Learning, Computer Science, Artificial Intelligence, TensorFlow, and Python communities."

"Emad Azad is a member of Python, Machine Learning, TensorFlow, Artificial Intelligence, and Computer Science communities."

> "Emad Azad is a member of TensorFlow, Machine Learning, Python, Computer Science, and Artificial Intelligence communities."

Generally, when a user is a member of n communities, one can construct $n!$ of these sentences. Aiming for the most valuable community and having equivalent sentences like above are sufficient to choose the CBOW over Skip-gram for community recommendation problem.

The rest of the section describes CBOW algorithm [35]. Where a text, such as the previous sentences, is defined as the bag of its words. Then by feeding this input to the model, a target word will be predicted.

The inputs of the model should be in the format of one-hot encoded vectors. The size of the vector, V, is equal to the number of unique vocabulary words. Every word has its unique vector with all zeros except the index associated with that word. This index is marked with one. The output format is also a one-hot encoded vector. An on-word context CBOW model is shown in Figure 5.3, where both the input and output have only one word [35].

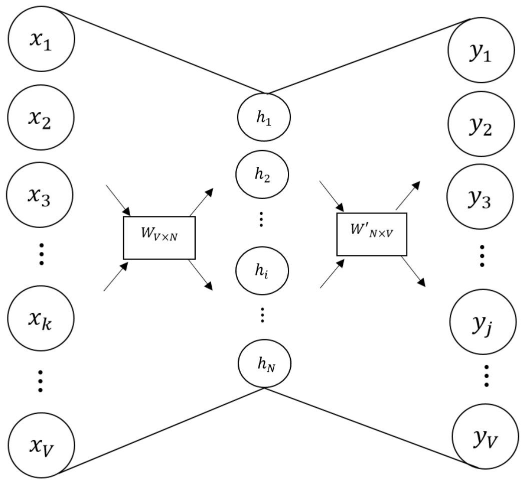


Fig. 5.3.: One-word Context CBOW Model

In this model, the input, X, is linked to the output layer by using two matrices. The first matrix, W, is called the input-hidden matrix. This $V \times N$ matrix projects the input to a hidden layer vector, h, of size N. Using Equation 5.1, h can be calculated without adding any activation function [35].

$$h = W^T X \tag{5.1}$$

Similarly, the second matrix, W', is of size $N \times V$ and is called the hidden-output matrix [35]. This matrix projects the hidden vector to a score vector, U, with Equation 5.2. The score vector is then passed through a softmax, a log-linear classification model, to find the output, Y.

$$U = W'^T h \tag{5.2}$$

Each element of U is refereed to as u_j. Then, the jth element of output, y_j is measured by Equation 5.3.

$$y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} \tag{5.3}$$

The object here is to maximize the probability of the output word, w_O, given the input word, w_I. Maximizing $p(w_O|w_I)$, is equivalent to maximizing the value of the index associated with the output word in Y. This index is labeled j^*. The loss function, E, is defined as $-\log p(w_0|w_I)$. Equation 5.4 describes how to calculate this loss function [35].

$$\begin{aligned} E &= -\log p(w_O|w_I) \\ &= u_{j^*} + \log \sum_{j'=1}^{V} \exp(u_{j'}) \end{aligned} \tag{5.4}$$

This loss function is used to update input-hidden and hidden-output matrices. The author in [35] explicitly explains how to do so. Before showing the procedures, two other variables need to be defined. One of which is the prediction error and is as follow:

$$e_j = \begin{cases} y_j & j \neq j^* \\ y_j - 1 & j = j^* \end{cases} \tag{5.5}$$

Then W' can be updated as:

$$v'^{(new)}_{w_j} = v'^{(old)}_{w_j} - \eta \cdot e_j \cdot h, \qquad \text{for } j = 1, ..., V \tag{5.6}$$

where $v^{'}_{w_j}$ is the jth column of W' and η is the learning rate.

The second variable is the derivative of E with respect to every hidden layer element, EH, an N-dim vector. Each of its units, EH_i, is the weighted sum of the prediction error of all words in the vocabulary. These weights are the element of W' [35].

$$EH_i = \sum_{j=1}^{V} e_j \cdot w'_{ij} \tag{5.7}$$

Where w'_{ij} is the entity in the ith row and jth column of W'. Now, the update rule for input-hidden matrix can be measured by Equation 5.8. Note that, in contrast to Equation 5.6, the equation is applied to only one row of W. This row, v_{w_I}, is the row associated with the input word, w_I [35]..

$$v^{(new)}_{w_I} = v^{(old)}_{w_I} - \eta \cdot e_j \cdot h \tag{5.8}$$

One can train the one-word context model with the help of Equations 5.1-5.8. Another CBOW model, multi-word context, is shown in Figure 5.4, where the input layer consists of multiply words. Instead of inputting a one-hot encoded vector to the model, C vectors are fed at once. The updating matrices are the same, except W is multiplied to the vector representation of every input word. The result is averaged in order to find the hidden layer vector. Thus, the Equation 5.1 changes to Equation 5.9, where X_i is the one-hot vector of the ith input word [35].

$$h = \frac{1}{C} W^T \left(X_1 + X_2 + \ldots + X_C\right) \tag{5.9}$$

The objective is still the same. That is, maximizing the probability of the output word, w_O, given the input words, $w_{I,1}, ..., w_{I,C}$. So, the goal is to maximize $p\left(w_O | w_{I1}, ..., w_{IC}\right)$. Consequently, the loss function will become:

$$
\begin{aligned}
E &= -\log p\left(w_O | w_{I1}, ..., w_{IC}\right) \\
&= u_{j^*} + \log \sum_{j'=1}^{V} \exp\left(u_{j'}\right)
\end{aligned}
\tag{5.10}
$$

where j^* is the index of the output word [35].

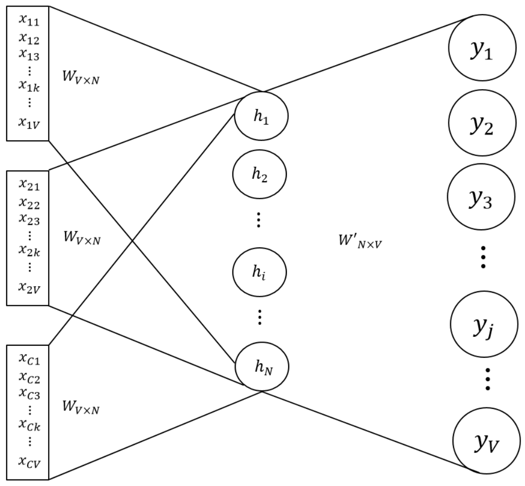


Fig. 5.4.: Multi-word Context CBOW Model

The way both of the matrices are updated stays almost the same. In fact, Equation 5.6 remains exactly unchanged. While Equation 5.8 should be repeated for all the words in the input layer, $w_{1,c}$, where $v_{w_{I,c}}$ is the row of W associated with the cth input word in the input layer.

$$
v_{w_{I,c}}^{(new)} = v_{w_{I,c}}^{(old)} - \frac{1}{C} \cdot \eta \cdot EH^T, \quad \text{for } c = 1, ..., C \tag{5.11}
$$

The hyperparameters of both models are the learning rate, η, and the number of units in the hidden layer, N. With the help of a validation set, the hyperparameters of the model can be tuned.

If a validation set is not used while training the models, the updating process should continue until the error does not change significantly. Otherwise, one can plot the error of the validation set per the number of epochs. Then stop the training when this plot plateaus. Note that, because after every iteration, only C rows of the input-hidden matrix are changed, the process might require many epochs.

5.4.1 Implementation

The CN dataset is modified to be fed into a multi-word context CBOW model. The communities of every user are considered as a sentence, and each community as a word. The same way mentioned in the previous section, except the first part, the punctuation, and the last word are removed. For the same example, "Emad Azad is a member of Computer Science, Python, Machine Learning, Artificial Intelligence, and TensorFlow communities.", the input words will be only the five communities that Emad has.

Each community is then represented by a one-hot encoded vector of size V, where V is the total number of distinct communities in the dataset. That is, the representation of each vector is all zero values except the index of that community, which is marked with one.

The model's inputs will be all the one-hot encoded vectors associated with a user's communities except one. That one community will be considered as the output. The idea behind this process is that if one of the user's communities is ignored, the other communities of that user should lead the model to find it as a recommendation because the user preferred that ignored community as they have already joined it.

The CN dataset is split into three sets to train the model; train, validation, and test set. For every user in the train and validation sets, the process mentioned is repeated as long as all of the communities are ignored once. For example, if somebody has four communities, this process continues four times. This way, there will be more data to train the model.

The model is implemented from scratch with the formula related to the multi-word context CBOW. The accuracy of the model will be addressed in Section 5.6.

5.5 Bidirectional Encoder Representations from Transformers (BERT)

Transfer learning is a new tool in recent years to solve the problems of insufficient data. The general assumption is that the train and test data should come from the same feature space or have the same distribution. In comparison, Transfer Learning enables using the trained model in one domain and transferring the knowledge to another field. For example, a network that can recognize pictures of apples is an excellent choice to be transferred. With a slight modification, it can then decipher images of pears this time [16].

Deep learning techniques are getting more attention from researchers recently [17]. These techniques require lots of data for the training part. Because not every domain can acquire many samples and information, transfer learning is prevalent in deep learning [17]. As a result, many pre-trained models in different disciplines are ready to be transferred to another field. A common approach is to add several layers on top of a pre-trained network. The rest of the model can then be fine-tuned with the little data available by freezing the initial layers. The fewer data available, the more layers are frozen.

Recent language representation models, consider the context either from left-to-right or right-to-left [18]. Whereas a Bidirectional Encoder Representations from Transformers (BERT) model is introduced in [18] that has obtained state-of-the-art accuracy for some natural language processing tasks. BERT looks at the whole context at once. This fact is similar to the idea of bag-of-words and can be utilized for the community recommendation problem. Moreover, [18] claims that BERT can be fine-tuned easily by adding exactly one additional output layer on top of the model.

5.5.1 Implementation

BERT has been trained on more than 3 billion word corpus. The way it was trained resembles the idea discussed in Section 5.4. In which a word was randomly masked, and the model objective was to predict it using only the context [18]. This is similar to removing a community from a user's community list and trying to predict it based on the other communities.

Using transfer learning on BERT for the community recommendation is an acceptable approach for three main reasons mentioned; the bidirectional aspect of BERT, the sparsity of CN dataset, and the ease of training BERT with few output layers.

The version of BERT used in this work has more than 100 million parameters, and a vocabulary that contains more than 30,000 tokens. These tokens convert a sentence to a vector to be fed to the model. The same procedure is taken for the CN data. Every sentence like "Emad Azad is a member of Computer Science, Python, Machine Learning, Artificial Intelligence, and TensorFlow communities," is converted to a vector of tokens. Note that one of the Computer Science, Python, Machine Learning, Artificial Intelligence, and TensorFlow communities is replaced with a masked token (['MASK']). In other words, the sentence is substituted as follow

> "Emad Azad is a member of Computer Science, Python, ['MASK'], Artificial Intelligence, and TensorFlow communities."

and then the new sentence is converted to a vector of tokens. This vector is the input of the new model, and the output will be the masked word, which in this example is Machine Learning. Also, this version of BERT is not case sensitive. That is, the letters of each word have to be transformed to lower case.

A softmax layer was added to the output layer of BERT to build the new recommender model. The size of this layer is equal to the size of unique communities in the system. This version of BERT's output returns a vector of size 768 for every token of the input. This information is then passed to the added layer. In the training part, all the BERT model is frozen, and only the parameters between the output of BERT and the softmax layer are tuned.

5.6 Results and Comparison

This section shows the accuracy of the three models explained before; NCF, multi-word context CBOW, and transferred BERT. Some figures are displayed to compare their result on the CN dataset.

There are many ways to compare the accuracy of a recommender system. Some of which are explained in Section 2.1. Here, two methods are defined, and the result of each for the three approaches are examined.

The first metric is "Top-5" accuracy, in which the label of every sample in the test set is compared to a list of recommendations with a length of five. If that label is in the list, that sample will have 100% accuracy; otherwise, it will be considered 0. Eventually, the average of all the sample's accuracy will be the final result. For example, assume the label for the following communities of a user is "Machine Learnin".

User 1: {Computer Science, Python, Artificial Intelligence, TensorFlow}

If "Machine Learning" exists in the recommendations list with a length of five, the algorithm will get 100% accuracy for this user. After performing the same process for all the users in the test set, the final accuracy will be the average of all the individual's accuracy.

The second metric is similar to "Top-5" accuracy and is called "Top-1." Their only difference is that the length of the recommendation list in this kind of measurement has to be equal to one. So, for the same example mentioned above, "Machine Learning" has to be predicted as the most probable community in order for the model to get 100% for that user.

5.6.1 Neighboring Collaborative Filtering (NCF)

As described in Algorithm 5 and Algorithm 6, the NCF model depends on the users' features to create the Neighboring. Here, two features are considered. One is the user's community, and the other is their Field of Study. The Top-1 and Top-5 accuracy are measured for both scenarios. Figure 5.5 shows these two metrics for different sizes of the train set and two sets of features.

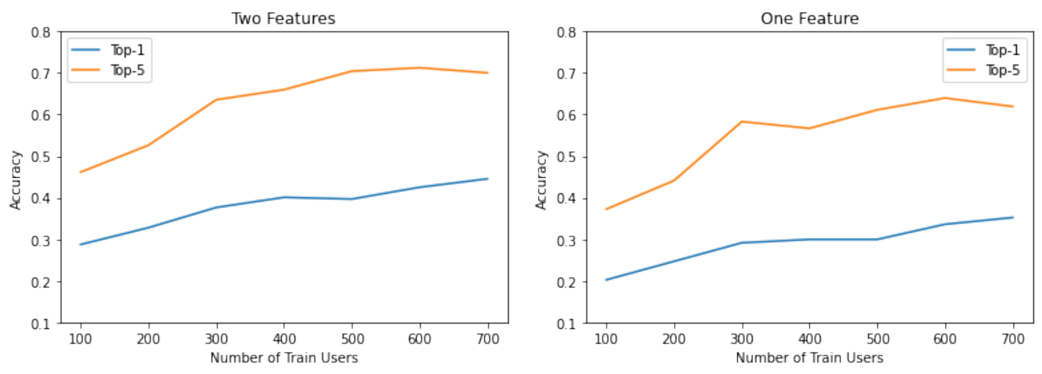


Fig. 5.5.: NCF Accuracy per Number of Train Users; NCF with Two Features (left) and NCF with One Feature (right)

As it is seen, as the number of users increases, the accuracy improves. Also, NCF works better when there are two features taken into consideration. The final Top-5 accuracy for NCF with two features is 70% and for one feature is 62%.

5.6.2 Multi-word Context CBOW

Many hidden layers and learning rates were examined and based on the validation set, two of which were chosen. One has six hidden layers, and the other eight hidden layers. In both cases, the learning rate of 0.01 is picked. Figures 5.6 and 5.7 show that the Top-5 and Top-1 accuracy plateau when the validation set's loss starts to flatten.

It is not common to use the test set in the training process. Here, this information is only used for illustration, and the final result is based on the epoch that training is stopped. For the model with six hidden units, the Top-1 and Top-5 accuracy are 24% and 46%, respectively. For the other model, Top-1 accuracy is 26%, and Top-5 accuracy is 50%.

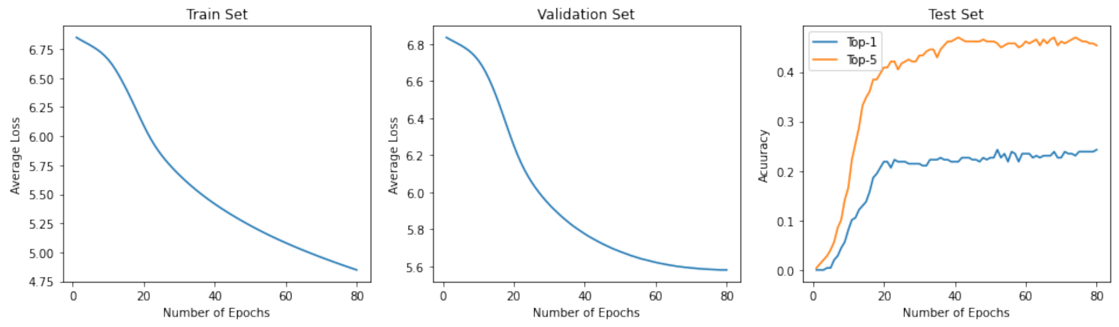


Fig. 5.6.: Train and Validation Losses along with the Top-1 & Top-5 Accuracy of CBOW Model with 8 Hidden Units

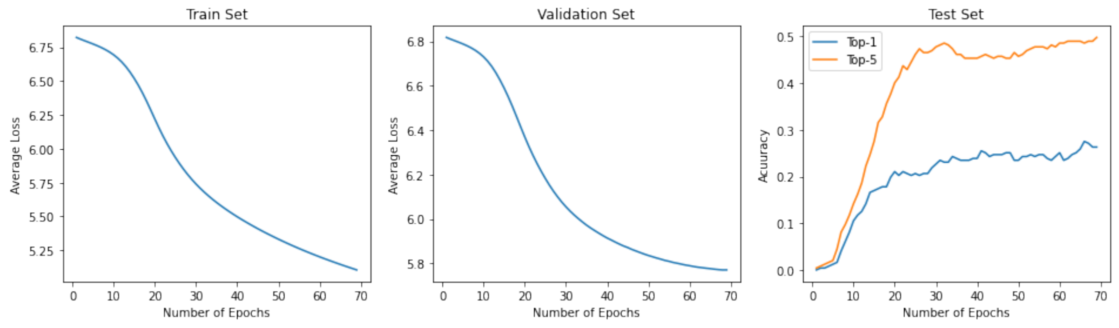


Fig. 5.7.: Train and Validation Losses along with the Top-1 & Top-5 Accuracy of CBOW Model with 6 Hidden Units

5.6.3 Transferred BERT

The new model explained in Section 5.5 is trained for 20 epochs. The summary of this model is shown in Figure 5.8. A dropout layer is also added to prevent over-fitting. After training the model with 84% Top-1 accuracy, the accuracy of the test set is measured. The Top-1 accuracy is 29%, and Top-5 accuracy is 55%. Both of these accuracies exceed both of the CBOW models.

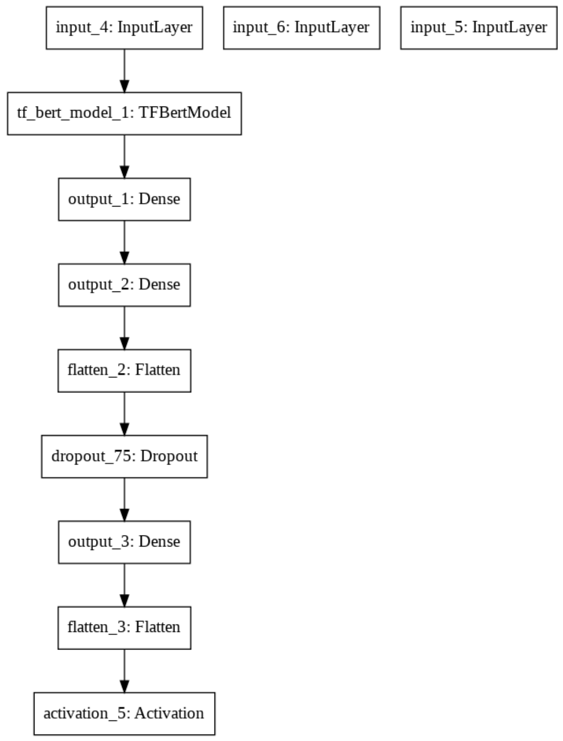


Fig. 5.8.: Model Summary of Transferred BERT

5.6.4 Comparison

Finally, the accuracy of all of these models is shown in Table 5.4. The NCF is only using the community feature to have a fair comparison, . Both CBOW models are presented here. After all, the NCF algorithm outperform all others.

Table 5.4.: Comparison of Top-1, Top-5, and Top-10 accuracy for different models

Accuracy	**Algorithms**			
	NCF	**CBOW (N=6)**	**CBOW (N=8)**	**BERT**
Top-1	34%	21%	21%	29%
Top-5	64%	42%	46%	55%
Top-10	77%	55%	57%	69%

5.7 Rumi Designs

The main goal of creating the user interfaces (UI) for community recommendations service was to have the best possible user experience (UX). Many designs were created to accomplish this goal. This section discusses the pros and cons of each.

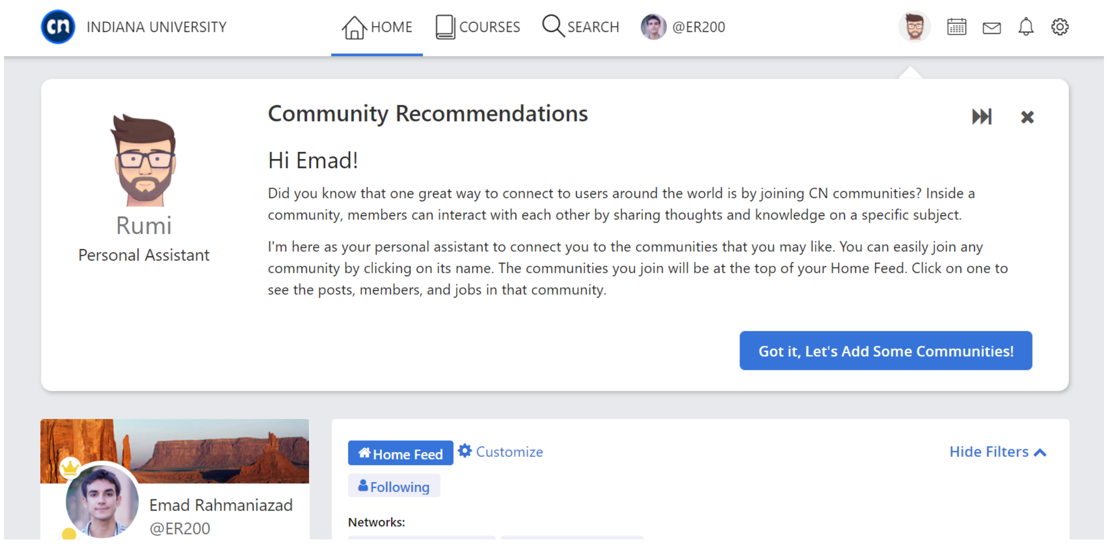
Before getting into the different ideas and sketches, one should be aware of all the functionalities of this service:

- Service Introduction: Rumi briefly explains the service
- Recommendations List: Rumi displays community recommendations based on the most appropriate to the least, and users can add them if they wanted
- Empty List: Rumi runs out of recommendations and asks for more information from the user
- Service Control: Rumi provides the ability to manually control the service by turning it on or off at any time

In general, based on which device the user uses, two separate UIs are developed. One is for the users who use a desktop to interact with Rumi, and the other is for mobile users. Therefore, every functionality will have two distinct designs.

5.7.1 Service Introduction

For Service Introduction, both mobile and desktop designs are almost the same. Figures 5.9 and 5.10 show the desktop and mobile view respectively. This UI is shown the first time that Rumi encounters a user with Community Recommendation service. As it is illustrated in the figures, he will concisely introduce the service.



Community Recommendations

Hi Emad!

Did you know that one great way to connect to users around the world is by joining CN communities? Inside a community, members can interact with each other by sharing thoughts and knowledge on a specific subject.

I'm here as your personal assistant to connect you to the communities that you may like. You can easily join any community by clicking on its name. The communities you join will be at the top of your Home Feed. Click on one to see the posts, members, and jobs in that community.

Fig. 5.9.: Community Recommendations Introduction - Desktop View

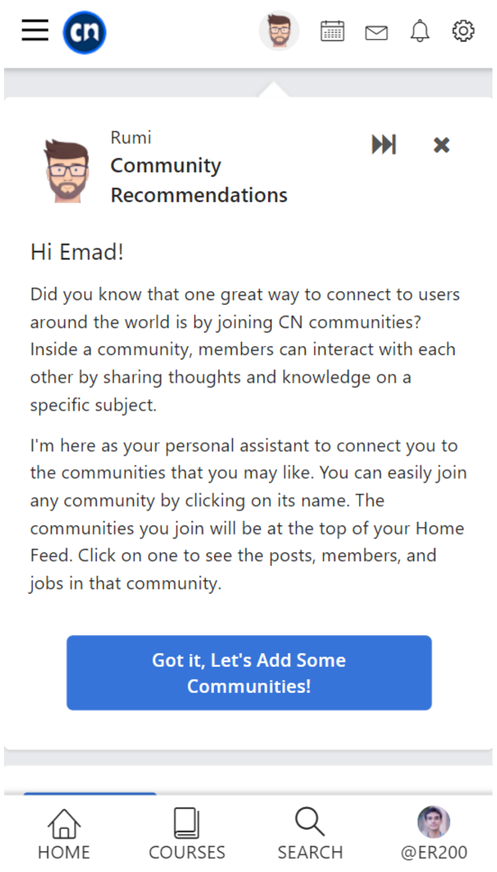
5.7.2 Recommendations List

After Service Introduction, the list of recommendations is shown. This is the UI that the users see the most, so several sketches were proposed before selecting the best version.

Desktop View

For the desktop view, the initial idea was to show three recommendations at a time, presented in Figure 5.11. Then ask users to join any of the recommendations by clicking on the “Add Community” button or discard them by clicking the “X” icon. The discard feedback could help Rumi to produce better recommendations. Also, an undo option was implemented to improve the user experience.

One disadvantage of this version is the existence of lots of white spaces between the recommendations. Hence, a more compact list was suggested in Figure 5.12. The new version also has a drawback; the buttons are too close to one another, making it hard for a user to select one.



Hi Emad!

Did you know that one great way to connect to users around the world is by joining CN communities? Inside a community, members can interact with each other by sharing thoughts and knowledge on a specific subject.

I'm here as your personal assistant to connect you to the communities that you may like. You can easily join any community by clicking on its name. The communities you join will be at the top of your Home Feed. Click on one to see the posts, members, and jobs in that community.

Fig. 5.10.: Community Recommendations Introduction - Mobile View

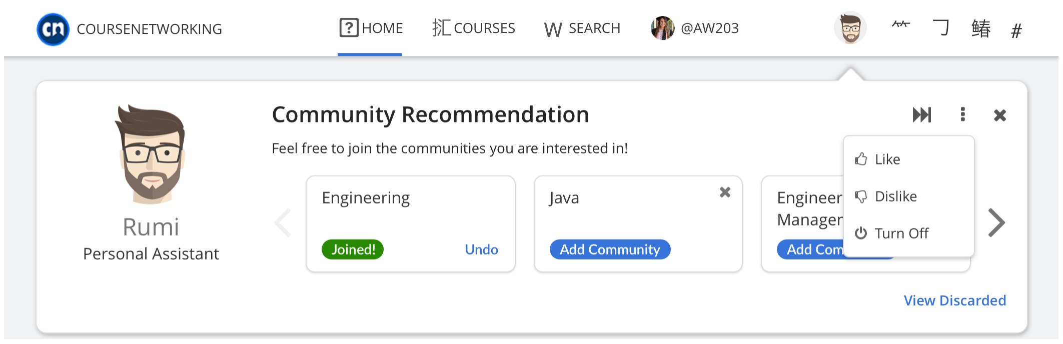
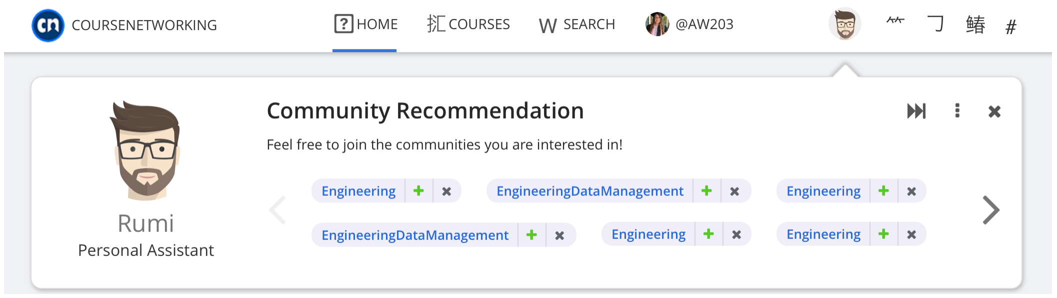


Fig. 5.11.: Initial Idea to Display Community Recommendations - Desktop View

Both of the previous versions enable users to remove a recommendation, but discarding something they do not own does not make sense. So, a final version was

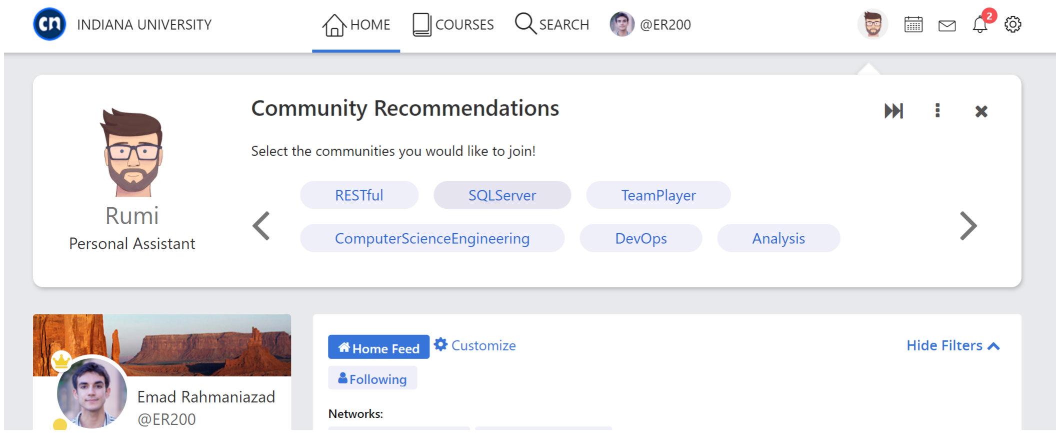


Community Recommendation

Feel free to join the communities you are interested in!

Fig. 5.12.: Compact Way to Display Community Recommendations - Desktop View

chosen that only allows users to add or undo their actions, refer to Figure 5.13. These actions are explained after announcing the final version for the mobile view.



Community Recommendations

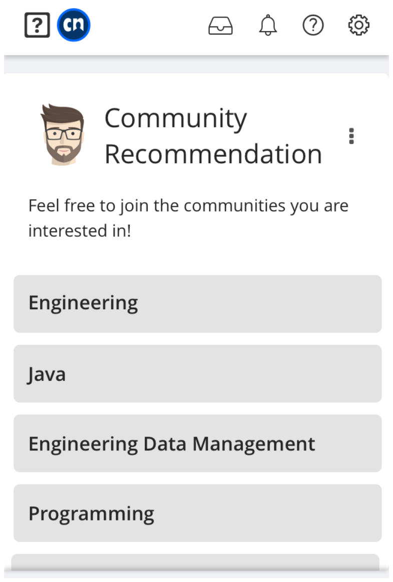
Select the communities you would like to join!

Fig. 5.13.: Final Design to Display Community Recommendations - Desktop View

Mobile View

On a mobile device, as shown in Figure 5.14, a list of recommendations are displayed in a way that the user has to scroll through them and find the ones that are of interest. To add or remove any of the communities, they have to slide the grey container to the right or left, respectively. Figure 5.15 demonstrates these actions.

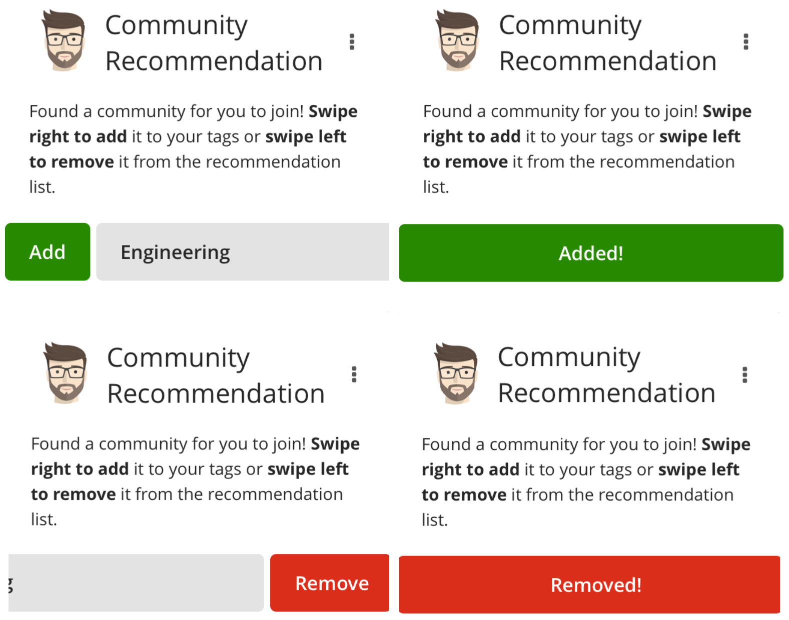
On mobile devices, the chances that someone accidentally touches the screen is high. Thus, providing an undo option is critical. This design did not have this option.



Community Recommendation

Feel free to join the communities you are interested in!

Fig. 5.14.: Initial Idea to Display Community Recommendations - Mobile View



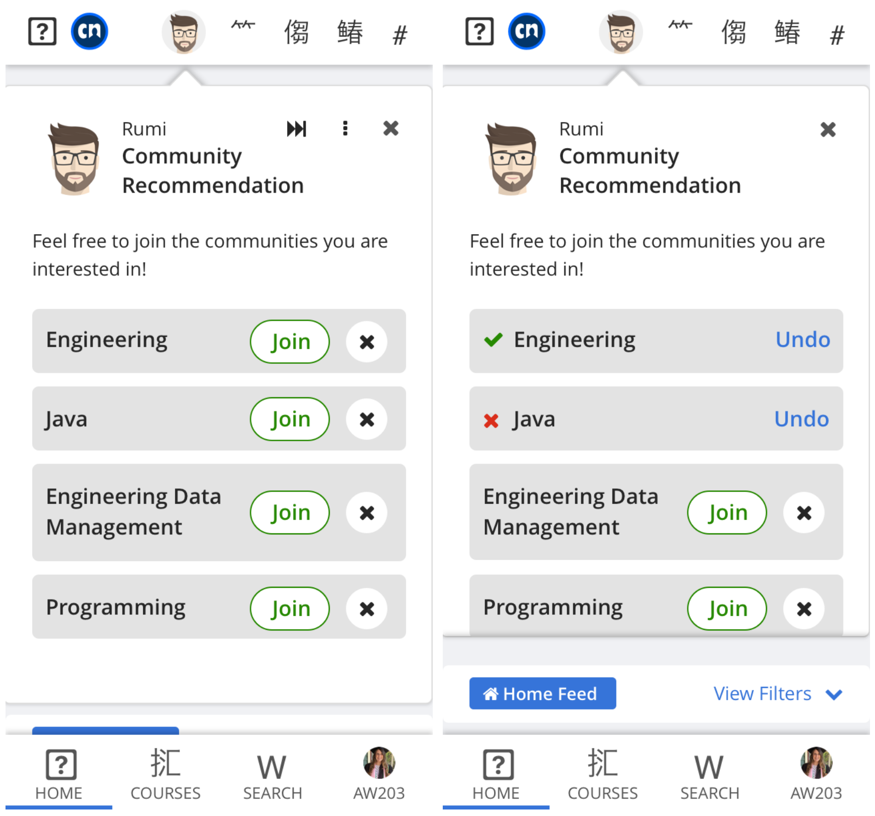
Community Recommendation

Found a community for you to join! **Swipe right to add** it to your tags or **swipe left to remove** it from the recommendation list.

Fig. 5.15.: Adding and Removing Community Recommendations - Mobile View

Moreover, on some operating systems, swiping right would change the URL and go to the previous page. Because of these two reasons, this version was a complete failure.

Buttons replaced the swiping option to overcome the two issues mentioned, exhibited in Figure 5.16. Nevertheless, similar to the desktop view, the discarding action was not logical.

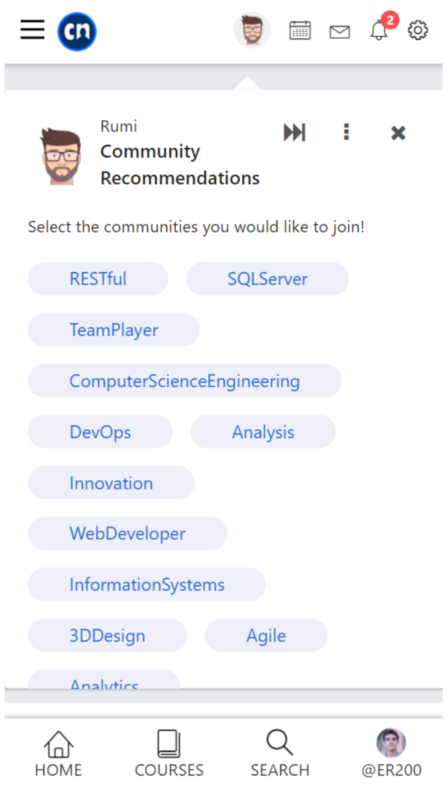

Rumi
Community Recommendation
Feel free to join the communities you are interested in!

Rumi
Community Recommendation
Feel free to join the communities you are interested in!

Fig. 5.16.: Unsuccessful Design for Community Recommendations - Mobile View

After all, the final mobile view is shown in Figure 5.17. It resemble the desktop view in Figure 5.13. Except, the user has to scroll through the recommendations, whereas on desktop, they had to clock on the left and right arrows to change between pages.

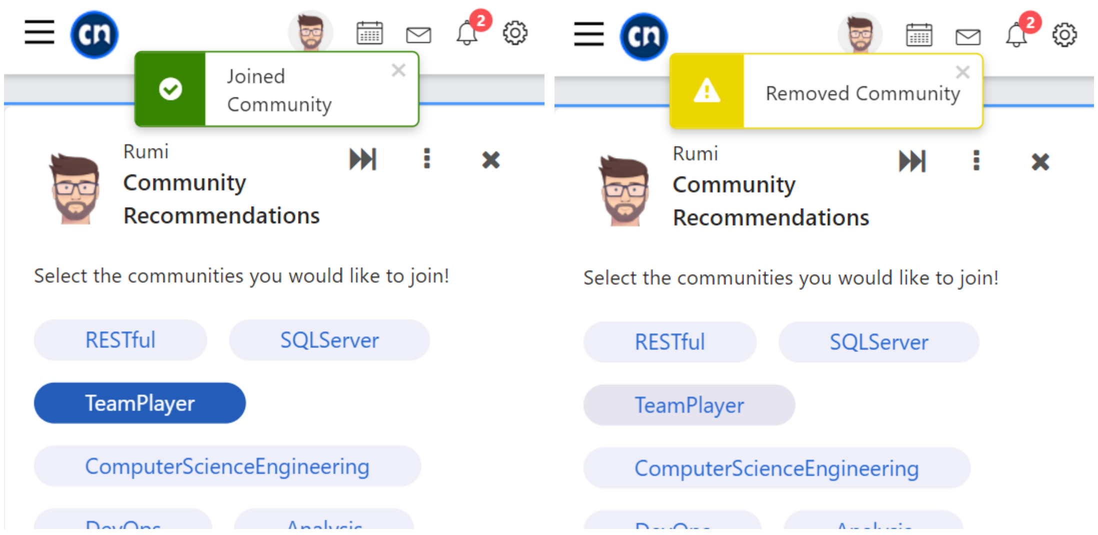
A notification is pushed when a community is selected, and the grey oval shape community becomes blue. The undo option is embedded in each oval. When the user selects any of the blue ovals, the action is undone, and another notification is sent. These notifications can be seen in Figure 5.18.


Rumi
Community Recommendations

Select the communities you would like to join!

RESTful
SQLServer
TeamPlayer
ComputerScienceEngineering
DevOps
Analysis
Innovation
WebDeveloper
InformationSystems
3DDesign
Agile
Analytics

Fig. 5.17.: Community Recommendations List - Mobile View


Joined Community

Rumi
Community Recommendations

Select the communities you would like to join!

RESTful
SQLServer
TeamPlayer
ComputerScienceEngineering

Removed Community

Rumi
Community Recommendations

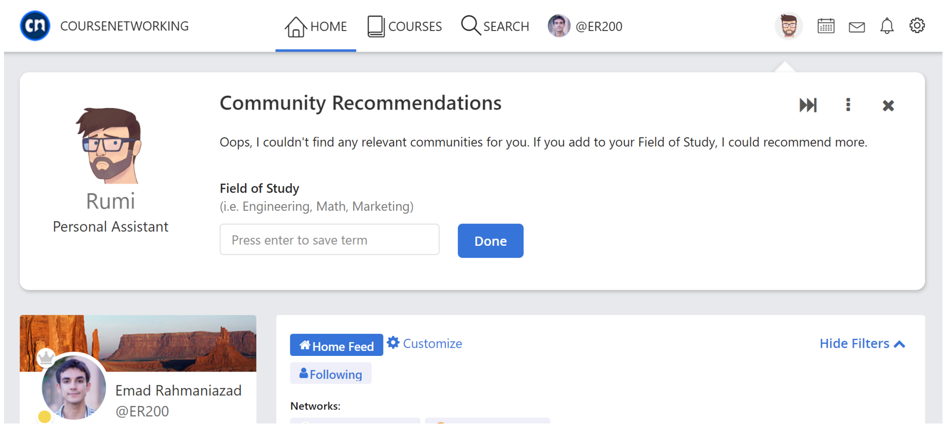
Select the communities you would like to join!

RESTful
SQLServer
TeamPlayer
ComputerScienceEngineering

Fig. 5.18.: Adding a Community or Undoing the Action - Mobile View

5.7.3 Empty List

Whenever Rumi runs out of recommendations, he tries to get more information from a user. Figures 5.19 and 5.20 shows how Rumi collects user's Field of Study to search for more recommendations. Also, Rumi has sad face here.



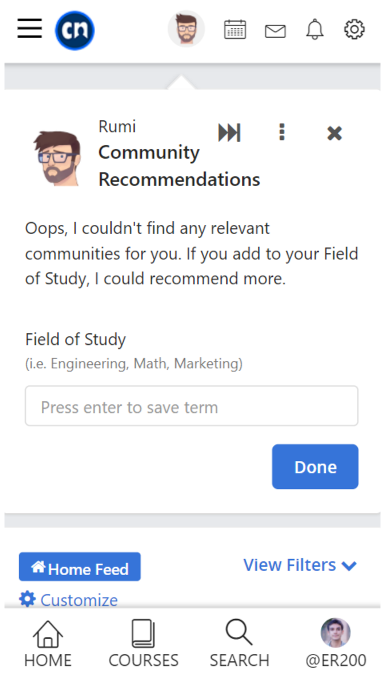
Community Recommendations

Oops, I couldn't find any relevant communities for you. If you add to your Field of Study, I could recommend more.

Field of Study
(i.e. Engineering, Math, Marketing)

Press enter to save term

Fig. 5.19.: Asking for User's Field of Study - Desktop View



Community Recommendations

Oops, I couldn't find any relevant communities for you. If you add to your Field of Study, I could recommend more.

Field of Study
(i.e. Engineering, Math, Marketing)

Press enter to save term

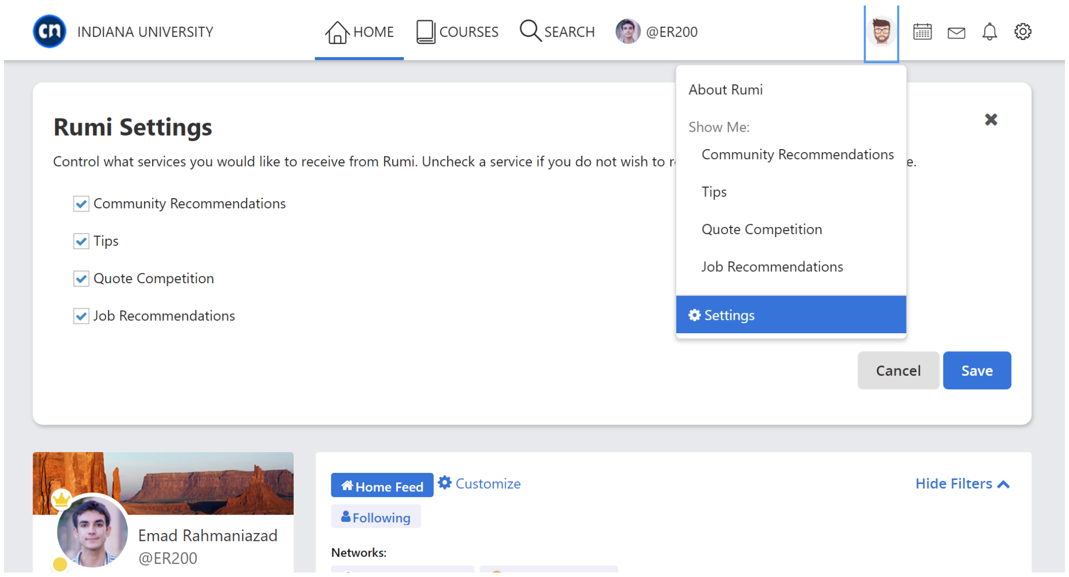
Fig. 5.20.: Asking for User's Field of Study - Mobile View

When users add to their Field of Study, Rumi will immediately suggest top communities of the typed in Field of Study.

5.7.4 Service Control

In the end, if the user gets bored of Community Recommendations, or they have already joined their desired communities, they can disable this service. A Settings page is provided for Rumi, shown in Figure 5.21. This page is located in the drop-down menu under the Rumi icon. With the help of this page, the user can turn a service on or off.

As displayed in Figure 5.22, a service can be turned off by unchecking the service's checkbox and saving. The mobile view looks similar to the desktop, so the figures are not included.

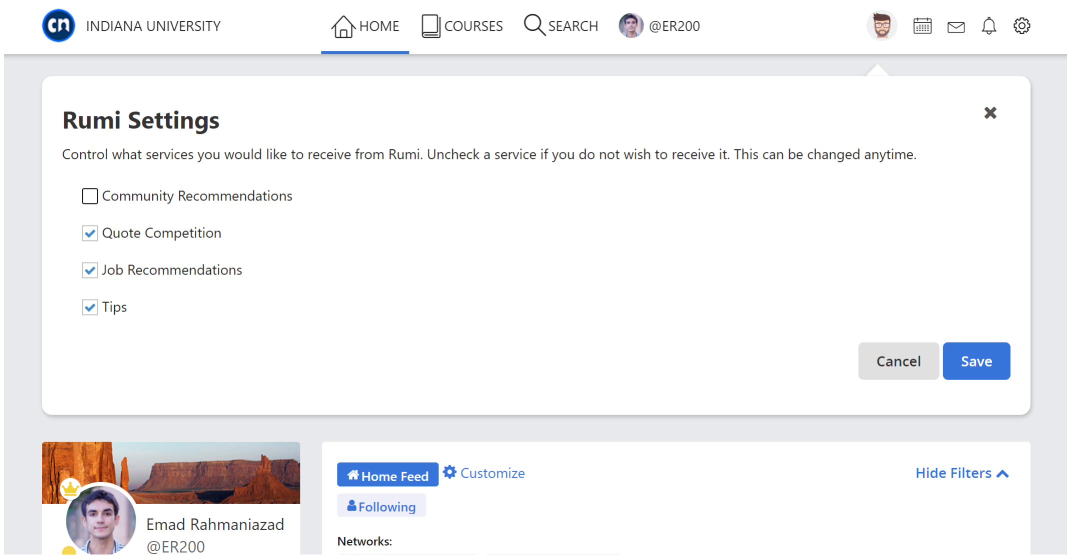


Rumi Settings

Control what services you would like to receive from Rumi. Uncheck a service if you do not wish to r… e.

- [✓] Community Recommendations
- [✓] Tips
- [✓] Quote Competition
- [✓] Job Recommendations

Fig. 5.21.: Settings Menu - Desktop View



Rumi Settings

Control what services you would like to receive from Rumi. Uncheck a service if you do not wish to receive it. This can be changed anytime.

- [ ] Community Recommendations
- [x] Quote Competition
- [x] Job Recommendations
- [x] Tips

Fig. 5.22.: Turning Community Recommendations off - Desktop View

6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this work, three different approaches are proposed, Collaborative Filtering (CF), Continuous Bag-of-words (CBOW), and Bidirectional Encoder Representations from Transformers (BERT). Each of these is modified in a way that it can generate community recommendations based on a user's profile.

A neighboring idea is added to the CF algorithm to change it to Neighboring Collaborative Filtering (NCF). Then a feedback mechanism is utilized to improve the performance of NCF. For the implementation of NCF, two separate algorithms, which both include the feedback structure, is explained. For the other two techniques, the data is translated into sentences to be fed into both models.

These algorithms' accuracy shows that NCF outperforms others in all of the metrics mentioned, such as Top-1 and Top-5 accuracy.

Moreover, the functionalities of Course Networking (CN) and the idea of an intelligent agent, Rumi, is discussed. One of the services that this agent provides is community recommendations, in which the NCF algorithm is used to recommend communities to CN users.

6.2 Future Work

The biggest challenges in this work are the sparsity and insufficiency of the CN dataset. Figure 5.5 shows the accuracy will improve if more data is provided. The NCF algorithm works better in this condition. The other approaches are data-driven, meaning the more data available, the higher the accuracy gets.

One other difficulty is that the communities of a user are not always related to each other. For example, if someone is interested in Python, Java, and Tennis and joins these communities, given Python and Java as an input, it does not make sense to predict Tennis as the output. A solution to this problem is to categories the communities into clusters.

This work can be further improved if more features are added to the NCF algorithm. Also, all the methods mentioned are using little information about every user. This information includes data that has a structure. For example, the CBOW and BERT model only use the communities of a user. No information from the user's profile nor their connection in the system is taken into consideration. The unstructured data, like user's posts or documents in the system, can be a useful source for improving the recommender system's accuracy.

Another drawback of this work is the lack of existence of a second database. CN dataset covers many community tags, but it does not convey the relationship between these communities. A good practice is to find a database that consists of users' skills or communities and run the algorithms mentioned in this work.

The inference engine described in this work can also be further improved. Currently, the time that an inference is called is set at 12 hours. Certainly, this time can be turned into unfixed time, which relies on the user's interaction. The rest of this section explains ideas for the new services that Rumi can offer. These ideas are as follow:

- Post Recommendations: There are machine learning techniques that can create new text out of previous information. They rely on transfer learning, where the model is trained on a massive database. Then the model is tuned based on the new input. The idea here is to use a pre-trained model that predicts sentences and paragraphs and fine-tuned it based on every user's posts to recommend new ones.

- Exam Recommendations: The same idea mentioned above can be applied here. Because most of the time, especially in 5-10 years, some instructors' exams or quizzes are usually the same. A model can mix and match them and change some metrics to create a new exam or quiz.

- Skill Recommendations: This service can employ the same algorithm that community recommendations use. The only differences are in the input data and the preprocessing pipeline.

- Job Recommendations: Redesigned Job Recommendations can make Rumi more productive. An engine that can be updated based on the user's feedback and improved the new recommendations' quality.

REFERENCES

REFERENCES

[1] M. Karimi, D. Jannach, and M. Jugovac, “News Recommender Systems - Survey and Roads Ahead.” Information Processing and Management, 2018.

[2] J. Itmazi and M. Megias. “Using recommendation systems in course management systems to recommend learning objects.” International Arab Journal of Information Technology, 5(3), 234-240. 2008.

[3] N. A. Alias and A. M. Zainuddin, “Innovation for better teaching and learning: Adopting the Learning Management System.” Malaysian online journal of instructional technology. Vol. 2, No.2, 27-40. 2005.

[4] F. Ricci, L. Rokach, B. Shapira, and K.B. P., “Recommender systems handbook,” Recommender Systems Handbook, 2011, pp. 1-35.

[5] A. Gunawardana and G. Shani, “A survey of accuracy evaluation metrics of recommender tasks,” Journal of Machine Learning Reearch 10, 2009, 2935-2962.

[6] J. Beel, S. Langer, M. Genzmehr, B. Gipp, and A. Nürnberger, “A Comparative Analysis of Offline and Online Evaluations and Discussion of Research Paper Recommender System Evaluation,” Proceedings of the Workshop on Reproducibility and Replication in Recommender Systems Evaluation (RepSys) at the ACM Recommender System Conference (RecSys), 2013.

[7] M. Bastian, M. Hayes, W. Vaughan, S. Shah, P. Skomoroch, H. Kim, S. Uryasev, and C. Lloyd. “Linkedin skills: large-scale topic extraction and inference.” In Proceedings of the 8th ACM Conference on Recommender systems, pages 1–8. ACM, 2014.

[8] G. Linden, B. Smith, and J. York, “Amazon. com recommendations: Item-to-item collaborative filtering,” IEEE Internet computing, no. 1, pp. 76–80, 2003.

[9] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-Based Collaborative Filtering Recommendation Algorithms,” Proceedings of the Tenth International Conference on World Wide Web - WWW, 2001.

[10] W. Chen, D. Zhang, and E. Chang. “Combinational collaborative filtering for personalized community recommendation.” In Y. Li, B. Liu, and S. Sarawagi, editors, Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 115–123. ACM, 2008.

[11] X. Han, L. Wang, R. Farahbakhsh, Á. Cuevas, R. Cuevas, and N. “Crespi, CSD: A multi-user similarity metric for community recommendation in online social networks,” Expert Systems with Applications, 53, 14–26. 2016.

[12] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space.” ICLR Workshop, 2013.

[13] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, “Distributed Representations of Words and Phrases and their Compositionality.” Accepted to NIPS 2013.

[14] M. Gutmann and A. Hyvarinen, “Noise-contrastive estimation: A new estimation principle for unnormalized statistical models.” In Proceedings of The Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS’10). 2010.

[15] B. Karlik and A. Vehbi, “Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks,” International Journal of Artificial Intelligence and Expert Systems (IJAE), vol. 1, no. 4, pp. 111–122, 2011.

[16] S. J. Pan, and Q. Yang, “A survey on transfer learning.” IEEE Transactions on Knowledge and Data Engineering, 22(10):1345–1359. 2010.

[17] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, “A survey on deep transfer learning,” 2018, arXiv:1808.01974, [online] Last Date Accessed: 2020-11-12.

[18] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, “Bert: Pretraining of deep bidirectional transformers for language understanding.” arXiv preprint arXiv:1810.04805, 2018, [online] Last Date Accessed: 2020-11-12.

[19] A. Jafari, “Conceptualizing intelligent agents for teaching and learning,” Educause Quarterly, vol. 25, no. 3, pp. 28–34, 2002.

[20] R. Agarwal, A. Deo, S. Das, “Intelligent agents in e-learning.” ACM SIGSOFT Software Engineering Notes, Vol. 29, No. 2, pp.1-3, 2004.

[21] P. Desharnais, J. Lu, T. Radhakrishnan, “Exploring agent support at the user interface in e-commerce applications,” International Journal on Digital Libraries, Vol. 3, No. 4, 284-290. 2002.

[22] M. Hosseini Asanjan, “Design and development of an intelligent online personal assistant in social learning management systems.” Master’s Thesis, Purdue University, Indiana University-Purdue University Indianapolis, 2019.

[23] “Coursenetworking, white paper,” https://www.thecn.com/aboutus, 2012, [online] Last Date Accessed: 2020-11-01.

[24] “IUPUI CyberLab,” https://cyberlab.iupui.edu, [online] Last Date Accessed: 2020-11-09.

[25] “Indeed,” https://www.indeed.com/, [online] Last Date Accessed: 2020-11-01.

[26] “Chegg Internships,” https://www.internships.com/, [online] Last Date Accessed: 2020-11-01.

[27] “HigherEdJobs,” https://www.higheredjobs.com/, [online] Last Date Accessed: 2020-11-01.

[28] A. Salamat, “Heterogeneous Graph-Based Neural Network for Social Recommendations with Balanced Random Walk Initialization” Master’s Thesis, Purdue University, Indiana University-Purdue University Indianapolis, 2020.

[29] “MongoDB Collections,” https://docs.mongodb.com/manual/core/databases-and-collections/#collections, [online] Last Date Accessed: 2020-11-04.

[30] “MongoDB Documents,” https://docs.mongodb.com/manual/core/document/, [online] Last Date Accessed: 2020-11-04.

[31] “BSON types,” https://docs.mongodb.com/manual/reference/bson-types/, [online] Last Date Accessed: 2020-11-04.

[32] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H.Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, “SciPy 1.0:Fundamental Algorithms for Scientific Computing in Python,” Nature Methods, vol. 17, pp. 261–272, 2020.

[33] W. Guo, H. Gao, J. Shi, B. Long, L. Zhang, B.-C. Chen, and D. Agarwal, “Deep natural language processing for search and recommender systems,” in Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD), Anchorage, AK, USA, Jul. 2019, pp. 3199–3200.

[34] Z. Fu, H. Gao, W. Guo, S. Kumar Jha, J. Jia, X. Liu, B. Long, J. Shi, S. Wang, and M. Zhou, “Deep Learning for Search and Recommender Systems in Practice.” In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD ’20). Association for Computing Machinery, New York, NY, USA, 3515–3516, 2020.

[35] X. Rong, “word2vec Parameter Learning Explained,” arXiv preprint arXiv:1411.2738. 2014, [online] Last Date Accessed: 2020-11-12.