DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2020

Graph convolutional network based friendship recommender system for a social network

HARSHA HOLENARASIPURA NARASANNA

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Graph convolutional network based friendship recommender system for a social network

HARSHA HOLENARASIPURA NARASANNA

Master in Information and Network Engineering
Date: December 1, 2020
Supervisor: Gustav Eje Henter
Examiner: Saikat Chatterjee
School of Electrical Engineering and Computer Science
Host company: GoFrendly AB
Company Supervisor: Erwan Lemonnier
Swedish title: Graf convolutional nätverksbaserat vänskap rekommenderar system för ett socialt nätverk

Abstract

This thesis aims to design and develop a deep neural network for friendship recommendations, and quantify its superior performance over a non-learned model for a given social network platform.

The given social network consists of a large number of users over 120,000. It is important to characterize each user to offer relevant services on the platform such as friendship recommendations. Some users fill their profile information while some do not. Some users have made friends while some are just new to the platform and have no friends. A good recommender system should be able to tackle such realistic cases by utilizing the available (although limited in few cases) information about each user. It must be able to incorporate both the user profile information as well as the user friendship connection information. It is challenging for traditional non-learned models to incorporate both these set of information and recommend well.

The field of deep learning has recently evidenced many innovations and advances and one of them is Graph Convolutional Networks (GCN). Lately, GCN-based models have proved their excellence in several recommendation tasks and shown state-of-the-art performance [1]. A Social network can be expressed as a graph with users as nodes and friendship connections as edges. Here, we examine the usage of a graph convolutional network to tackle the recommendation problem. Also, this thesis employs state-of-the-art pre-trained language model S-BERT for semantic representation of user textual data. This thesis investigates the viability of GCN-based models for friendship recommendations for individual social network users.

In particular, a GCN model shall be developed based on PinSage architecture [1]. Then the GCN models shall be evaluated for three different setups. The obtained results show that the GCN-based model outperforms the non-learned model by a definitive margin. In particular, the findings show that a GCN model trained on a single city with rich user features yields the best results. This exhibits the versatility of GCN and corroborates its recent success in recommendation systems. This work demonstrates the rich characterization of each node in the graph (or user), which not only can be used to offer friendship recommendations but also to recommend other entities on the platform.

Sammanfattning

Detta examensarbete ämnar kvantifiera till vilken grad ett djupt neuralt nätverk är bättre än en icke-inlärd modell på att rekommendera potentiella nya vänner till individer på ett socialt nätverk.

Ett socialt nätverk består typiskt av ett stort antal individuella användare. Det är viktigt att karaktärisera varje användare för att erbjuda dem relevant service, såsom rekommenderade vänner. Vissa användare fyller i profilinformation, medan andra inte gör det. Vissa användare har vänner i nätverket, medan anda är nya på plattformen och saknar vänner där. Ett bra rekommendationssystem skall kunna hantera båda dessa fall, vilket inkluderar att analysera både användares profilinformation och deras väncirklar. Det är utmanande för traditionella, heuristiska metoder att inkorporera båda dessa informationskällor och rekommendera väl.

Djupinlärning har på senare tid uppvisat många innovationer och framsteg, bland dem konvolutionsnät på grafer (Graph Convolutional Networks, GCN) [1]. GCN-baserade modeller har visat sin styrka genom ledande resultat i flera rekommendationssystem. Ett socialt nätverk kan formuleras som en graf där noderna är användare och kanterna är anknytningarna dem emellan. Detta gör det möjligt att använda konvolutionsnät på grafer på problemet. Detta examensarbete använder också den ledande förtränade textmodellen S-BERT för att extrahera särdrag ur textdata från användarna. Examensarbetet undersöker potentialen hos GCN-baserade modeller för att rekommendera nya vänner till individuella användare.

Mer specifikt utvecklar vi en GCN-modell baserad på en arkitektur kallad PinSage. Vi studerar GCN-modeller för tre olika fall. I alla tre situationerna visar resultaten att GCN-baserade modeller med marginal ger bättre resultat en icke-inlärda modeller. Våra resultat visar särskilt att en GCN-modell tränad på en enskild stad med rika användarsärdrag ger bäst resultat. Detta visar flexibiliteten hos GCN:er och överensstämmer med deras framgångar i andra rekommendationssystem. Examensarbetet demonstrerar rika användarsärdrag som inte bara kan möjliggöra vänskapsrekommendationer utan även kan rekommendera annat innehåll på plattformen.

Acknowledgements

This thesis would not be complete without the support of many people, and a big heartfelt thank you for everyone. I sincerely want to thank each and everyone who has been part of my thesis and my life during the past two years. I might have forgotten to name some of you here for which I apologize in advance.

Big thanks to ‘GoFrendly AB’ (or just GoFrendly) for providing me with the opportunity to work on this challenging project, Mr. Erwan Lemonnier, CTO of GoFrendly, and Ms. Julia Sporre, CPO of GoFrendly, for the regular guidance, be it technical or moral, as well as all the other employees who were a great company to work and interact with. I would like to appreciate the great work environment and the wonderful learning experience that was provided.

I would like to thank my supervisor, Mr. Gustav Eje Henter, associate professor in the division of speech, music and hearing at the School of EECS at KTH, for major supervision provided during this period. You have been a great inspiration from whom I learnt a great deal. The interactive sessions we had were eye-opening and provided me with immense knowledge and directions in which I could drive this thesis forward. The attention to detail, the lessons from your rich academic experiences paired with no urgency was like icing on the cake, just made this work even more enjoyable.

I want to thank my examiner Mr. Saikat Chatterjee, associate professor in the division of information science and engineering at KTH, for all the assistance and encouragement provided during this period. You have been a great influence on my interest in machine learning. I have learnt the principles of machine learning from your course lectures, which is foundational knowledge for this thesis work.

I am grateful to all my family and friends, who have always been there for me, in more possible ways than I could ever wish for. Maneesh, Bharathesh, Suhas and many others whom I could not mention, thank you one and all from the bottom of my heart.

Last but most importantly, a huge thank you to my wonderful parents, Mr. Narasanna and Mrs. Uma and my beloved sister Varalakshmi for all the love, support and encouragement, without whom I would not be where I am today.

Contents

Chapter 1

Introduction

The goal of this thesis project is to develop an effective prediction model for producing individual friendship recommendations for the given social network platform based on user profile information and friendship connections.

A social network is a social structure which facilitates communication between a group of people who are related such as by common interests, shared values and friendship connection [2]. A social network platform (or social networking service) in the context of this thesis is an online service, which people use to build social networks or social relationships [3]. An unprecedented number of users have signed-up for online social networking services. The success of online social networking services can be seen in their prevalence in our society today, with over millions of active users worldwide [3]. In fact, the number of social network users is projected to reach nearly 3.09 billion worldwide by the end of 2021 [4]. In particular, to discover friends and to socialize with them are the key social dynamics, which a social network platform must facilitate.

Over-choice is a cognitive impairment in which people have a difficult time making a decision when faced with many options. With the ever-growing volume of information, recommendation systems, in general, have been an effective strategy to overcome the information over-choice. Recommendation systems are usually a secret recipe for the service providers. Their ability to elicit the right information from a vast pool of information has led to its widespread adoption in many internet applications [5]. In fact, over 80% of the videos watched on Netflix is through recommendations [6] and 60% on YouTube [7]. Netflix estimated that its recommendation engine is worth a yearly $1billion

[8]. Clearly, there are several benefits that businesses can reap using recommendation systems.

For a social network with large userbase such as over 100,000 users, the recommendation system is vital to ensure the user's over-choice is addressed. However, not only the platform users grow in number but also the social dynamics of these users evolve. Thus, it is imperative to employ modern high-performance methods to improve the recommendations so to facilitate the key social dynamics of friendship formation.

1.1 Problem description

The social network platform for our case is 'GoFrendly', a female-only social networking platform with over 100,000 users spread in many cities across the globe[1]. To find and meet female friends is one of the top interests of the users. Users voluntarily sign-up and fill-up their profile information.

User profile information includes data such as *my_story, i_am, meet_for, marital_status, kids, age* and *location*. The total number of friendship connections (two-way friends) on the platform is around 120,000, which is equivalent to one per user. The reason for such a low number might be because the platform allows one-way added friends to interact without becoming explicit friends (two-way friends). The total blocked user pairs come around 13,684. Lastly, the friendship connection is seen as a positive social relationship while blocked user pair is seen as a negative social relationship.

Our goal is to offer personalized friendship recommendations to individual users. This problem falls under semi-supervised learning because the social relationship between any pair of users is either labelled (positive or negative, as defined above) or unlabelled as not-known (or not-yet seen).

1.2 Research question

The research question that has been investigated is:

'Can a suitable scalable deep neural network perform better than a non-learned friendship recommender for a given social network, and, if so, by how

[1]www.gofrendly.se

much?'.

A non-learned friendship recommender is the current recommender system at goFrendly, which for a chosen user shortlists user set with a similar profile. Then, the user set is filtered using user preferences such as *age* and ranks the users based on heuristic factors such as last active users, completed profile and not seen before users.

Figure 1.1 shows the high-level flow of the friendship recommendation system. The non-learned system is highlighted in the yellow box, which simply shortlists the similar user profiles whereas the deep neural network, which shall be developed, is highlighted in the orange box. The scope of the thesis involves a comparison between these two models i.e., non-learned system vs. deep neural network model for the task of friendship recommendations. The grey box titled 'Heuristic Rank Factors' is optional with Deep Neural Networks (DNN). Although DNNs are capable of delivering a ranked list, heuristic-based ranking can be considered if the ranking ability of DNN is not satisfactory.

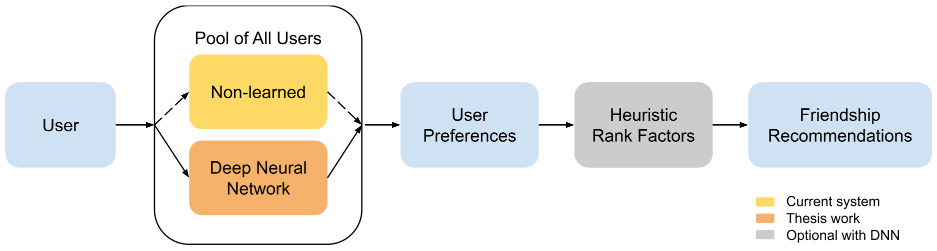


Figure 1.1: Main flow of friendship recommendation system

1.3 Report structure

This thesis is organized as follows. Chapter 2 provides prerequisite knowledge, foundational concepts and a formal literature review. Next, Chapter 3 introduces the choice of technical and practical methods, and its implementation details involved in tackling this problem. In Chapter 4, we benchmark the performance of recommender systems, make comparisons and highlight our findings. It also covers experiments that were conducted and analysis of its results. Lastly, Chapter 5 consists of discussion and conclusions encompassing the overall outcome of the project, practical usefulness and its social & ethical implications. In the end, a section is devoted to describing the scope for its future work and concludes with a bibliography.

Chapter 2

Background

2.1 About GoFrendly

GoFrendly[1] is an online social networking service exclusively for women. The company was founded in 2016 in Stockholm, Sweden. The word ‘frend’ stands for a person whom you do not know in real life, yet is a friend on the internet[2]. As of 2020, GoFrendly is a free mobile application and available to the public on Android and iOS devices on selected countries including Sweden, Germany, Norway and India. GoFrendly enables users to find, meet and make friends at ease. GoFrendly tries to connect with like-minded friends in one’s local area and create meaningful friendship connections. Furthermore, it features a personalized matching feed based on one’s interests and life situation. It also features the discovery of local activities such as book reading clubs, cooking workshops, dinner parties or a girls’ night out, and meet new women friends who share interests in music, travel, food, arts, yoga, books, photography, movies, hiking and more.

Users can sign-up to create an account on the mobile application at no cost and then complete their user profile with information such as *my_story, i_am, meet_for, marital_status, kids, age* and *location*. *Age* and *location* are numerical data whereas *my_story* is textual data, and lastly, *i_am, meet_for, marital_status* and *kids* are categorical data chosen from a list. Then, a user can find friends on the platform and meet them in real life. Besides, the application offers many more features to elevate the social experience such as in-app chat service, video call service, social activity finder, activity calendar,

[1]www.gofrendly.se

[2]https://www.urbandictionary.com/define.php?term=Frend

and other interesting features constantly being rolled out for its users.

2.2 Recommendation systems

2.2.1 Common recommendation system types

Recommendation models are mainly classified into three categories: collaborative filtering, content-based, and hybrid recommender system [5]. Firstly, the collaborative filtering makes recommendations by learning from the user's historical interactions; for example, recommending new friends based on current friends circle. However, this mechanism suffers from a cold start problem for new or isolated users wherein the system lacks sufficient information about the user's friend circle to make accurate predictions. Secondly, the content-based recommendation is based primarily on information that users have described themselves [5]. Although this solves the cold start problem, it suffers for incomplete profiles, i.e., the ones with little or no data of the description about themselves. Finally, a hybrid model refers to a recommender system that integrates two or more types of recommendation strategies [5]. The hybrid systems generally provide superior performance and thus, the choice for our case [5].

2.2.2 Deep learning based recommender systems

This section outlines the reasons for applying deep learning techniques to recommendation systems. Deep learning is generally considered to be a sub-field of machine learning. The typical defining essence of deep learning is that it learns deep representations, i.e., learning multiple levels of representations and abstractions from data. One of the most attractive properties of deep neural architectures is that they are (1) end-to-end differentiable and (2) provide suitable inductive biases catered to the input data type [5]. Further, the strengths of deep learning-based recommender systems include (1) nonlinear transformation which enables them to capture complex and intricate user interaction patterns, (2) representation learning which eliminates the expensive feature engineering and enables the models to include heterogeneous content such as numerical, categorical and textual information, and (3) flexibility which comes with the freely available deep learning frameworks such as PyTorch[3] which enables efficient and modularized development [5].

[3]www.pytorch.org

However, these models do have potential limitations. (1) interpretability: it is difficult to interpret the activations and the weights of deep neural network models. This limits the explainability of learned models [5]. (2) data requirement: deep learning models require a large volume of data to adequately cater to its rich parameterization. (3) extensive hyperparameter tuning involved in the deep neural networks. However, these concerns have been mostly addressed with the advent of modern deep neural networks [5].

Moreover, deep neural networks offer a vast set of architectures and mechanisms to tackle various recommendation problems. Some of the architectures are as follows: Multi-Layer Perceptron (MLP), autoencoders, CNNs, RNNs, RBM, Neural Autoregressive Distribution Estimation (NADE), neural attention models, adversary networks, Deep Reinforcement Learning (DRL), Graph Neural Networks (GNN) [5][1]. For all these reasons deep neural network-based solution is the first choice of solution for recommendations.

2.3 Neural networks

2.3.1 Types of learning problems

There are four main types of learning problems in neural networks: supervised, unsupervised, semi-supervised and reinforcement learning. Generally, a learning problem is categorized based on available data and the end goal.

Supervised learning describes a class of problem that involves using a model to learn a mapping between input examples and its target variable. Here the data consist of input values with definite output labels for each sample. Classification and regression are the common types of problems that are solved by supervised learning methods. On the other hand, unsupervised learning involves using a model for unlabelled data (target variables are not known) to identify patterns and structures present in them. The goal in such unsupervised learning problems may be to discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to lower dimensions.

Next, reinforcement learning describes a class of problem that aims at using observations gathered from the interaction with the environment to take actions that would maximize the reward or minimize the risk. Common algorithms include Q-Learning [9], Temporal Difference (TD) [10], Deep Adversarial Networks [11]. Lastly, Semi-supervised learning describes a problem wherein some portion of the available data is labelled while the rest is unlabeled for training as well as testing. Semi-supervised learning falls between unsupervised learning (with unlabelled training data) and supervised learning (with only labelled training data). Friendship recommendations can be seen as a semi-supervised learning problem as the social relationship between any two parties is either labelled as friends or blocked user pairs, or unlabelled.

2.4 Literature review

2.4.1 How the literature review was conducted

Google Scholar was used as the main search engine. Search keywords such as deep learning, social network, friendship recommendation, link prediction and other related specific words in combination were used. Relevant high profile conference publications such as the proceedings of KDD and ECIR were screened and the ones which are peer-reviewed and having significant citations were preferred. Popular research projects such as the Stanford Network Analysis Project (SNAP) [12] were considered. In addition, social network service providers - such as Facebook and LinkedIn - recommendation strategies were referred. Besides, several research works have been studied but omitted from this report to ensure brevity and relevance.

2.4.2 Industrial published recommendation methods

Friendship recommendation is one of the key features of many social network platforms. Recommendation strategies are a secret recipe of the companies, but information on some of the methods in use is publicly made available. Social network giants namely LinkedIn and Facebook were considered to examine their methods. However, the performance of their friendship recommendation is not publicly quoted.

LinkedIn network recommendations

LinkedIn suggests new network connections under the 'People You May Know' feature based on common profile information. Another feature called 'People Also Viewed' shows the member profiles that the viewers of one's kind have also looked at [13]. The former is a content-based approach while the latter is collaborative. More technical detail has not been provided.

Facebook Network Recommendations

Facebook's 'People You May Know' feature recommends new friends and these suggestions are based on the factors such having friends in common, from being in the same Facebook group or school or work network and lastly, the contacts that have been uploaded by the user [14]. These quoted methods are non-learned and heuristic. Besides, finding new friends is not the top reason users use Facebook.

2.4.3 Academia

Stanford Network Analysis Project (SNAP)

SNAP is an active network analysis research project led by Prof. Jure Leskovec and has highly cited publications. More recent ones involve deep neural network-based network analysis methods [1]. Alongside, it publicly hosts several databases related to network analysis and a general-purpose scalable network analysis library [12].

2.4.4 Referred conference publications

In the literature review, several research works were considered and below lists relevant popular conferences which were reviewed.

1. SIGIR: Special Interest Group on Information Retrieval.
2. KDD: Knowledge Discovery in Databases.
3. ECIR: European Conference on Information Retrieval.
4. WWW: International World Wide Web Conference.
5. ICTIR: International Conference on Theory of Information Retrieval.

2.4.5 Main idea

A social network can be represented as a graph with the user as node and friendship connections as edges (user and node shall be used interchangeably hereafter). For a non-isolated node, its connection with its neighbour nodes can be expressed in terms of k-degree connections where k can take positive integer values. For example, 1^{st} degree connections are the nodes which are reachable in 1-hop i.e., the people who are directly connected. 2^{nd} degree connections are nodes reachable in 2-hops and similarly applies to k-degree connections. Figure **??** demonstrates the graph for a sample social network.

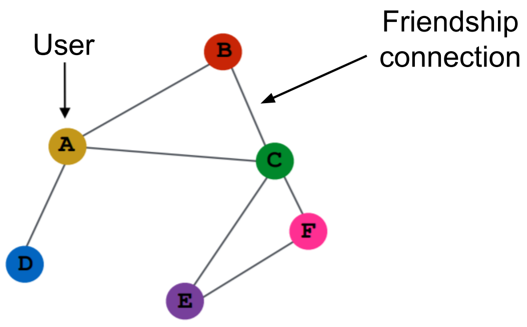


Figure 2.1: An example of social network graph
(Adapted from Figure 1 of [1]).

Our task can be subdivided into two:

1. User Characterization: To characterize each user based on profile information and friends circle.
2. Encoding: To find a pattern between user characteristics and friendship connections.

User Characterization

A rich user characteristics need to include (1) user profile information (node information) (2) user friend circle (graph structure information or simply graph information). For this case, a model can be classified based on information used. Table 2.1 shows the classification of the models. A random recommender would mean none of the user-specific information is used. A non-learned model in this case uses only the user profile information. A Deep

Classification based on information used	No node information (No user profile info.)	Node information (User profile info.)
No graph information	Random recommender	Non-learned model
Graph information (k-degree connections)	DNN model with randomly initialized node information	DNN model

Table 2.1: Model classification based on information used.

Neural Network (DNN) for this case must be capable of incorporating both node information and graph information up to k-degrees. DNN shall be developed as a part of this thesis work. The focus of this research is mainly on the non-learned model and the DNN model.

To characterize a user, the user's information needs to be represented in a mathematical form for further processing. Firstly, we identify relevant methods to mathematically represent user profile information. Secondly, we need to identify relevant methods to mathematically represent the local neighbourhood for any given node. Both of these act as pointers for our literature study.

Encoders

The user characteristics are encoded such that the relevance between users characteristics can be associated with friendship compatibility. In particular, we encode each user so that the similarity in a mathematical embedding vector space approximates friendship compatibility in the original network [15].

The encoder is trained using an existing set of friendship connections. Thereafter, all the users are encoded onto the embedding vector space called as user/node embeddings. User embeddings are distinct numerical vectors that are tagged with each user respectively, which represent its friendship compatibility position in the embedding vector space. Thereafter, the nearest neighbour search for any target user in the embedding space forms a list of potential friendship recommendations. Figure 2.2 shows the idea of the described encoder.

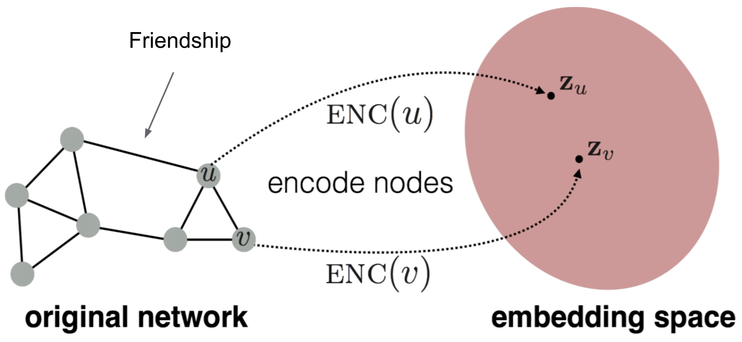


Figure 2.2: Intuition behind an encoder (Adapted from Figure 2 of [15])

The next section follows a formal literature review towards finding relevant solutions for the aforementioned tasks: user characterization and encoding. The user characterization incorporates both user profile information and user-friend circle information. Firstly, the user profile needs to be mathematically represented for further processing and the methods are described in 2.4.6. Secondly, the user-friend circle also needs to be mathematically represented and the methods are described in 2.4.7. Thirdly, the user characteristics need to be encoded corresponding to its friendship compatibility position. Interestingly, the representation of user friend circle and encoding of user characteristics can both be obtained through a single set of methods and are described in 2.4.7. Then, we obtain user/node embedding for each user. Finally, the friendship recommendations from node embeddings are described in 2.4.8.

2.4.6 Representation of user profile information

The goal is to represent user profile information in a mathematical vector form and shall be called and identified as user features here onwards. GoFrendly user profile information consists of *my_story, i_am, meet_for, marital_status, kids, age* and *location*. *Age* and *location* are numerical data and shall be kept as they are whose vector size is 1 and 2 respectively. Categorical data are *i_am, meet_for, marital_status, kids*, which are options chosen from a set of options. Categorical data are one-hot encoded whose total vector size is 45. Figure 2.2 shows the breakdown. Furthermore, *my_story* is a textual data whose semantics need to be considered for our case. Now, the focus is on the semantic representation of the user's my_story.

Several advancements have taken place in the last decade in the field of semantic representation of textual data. Various word embeddings mechanisms have been proposed. Introduction of word vectors involving Continuous BoW (CBOW) and SkipGram architectures [16] were the early success in the semantic representation of words. Variants of the word vectors are as follows: word2vec from Google [17], GloVe by Stanford [18], and fasttext by Facebook [19], all of which have shown similar performances. Further step-up in the performance were seen in node2vec architecture, which is based on a biased random walk wherein the words are represented in the graph [20].

More recently, language transformers [21] have profoundly improved the estimation of textual semantics viz., BERT [22] and RoBERTa [23] have set the new state-of-the-art performance on the semantics of textual data. However, these models involve massive computational overhead in the training setup when run on large-scale datasets. A more recent publication titled Sentence-BERT [24], a modification of the pre-trained BERT network offers comparable accuracy with significantly fewer computations at inference. In the next section, we introduce the technical details of the SBERT mechanism.

Sentence-BERT Mechanism

Sentence-BERT (SBERT) is a modification of the pre-trained BERT network that uses siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using similarity metrics such as cosine-similarity. This reduces the effort for finding the most similar pair from 65 hours with BERT or RoBERTa to about 5 seconds with SBERT, while maintaining the accuracy from BERT [24]. A large disadvantage of the BERT network structure is that no independent sentence embeddings are computed, which makes it difficult to derive sentence embeddings from BERT. To bypass these limitations, SBERT proposes that single sentences are passed through BERT and then employ a pooling layer to derive a fixed-sized embedding which then trained in a siamese structure. The parameters of the BERT model are fine-tuned such that the semantically similar sentences are cosine similar in the mathematical vector space. Figure 2.3 shows the SBERT architecture. This model is when evaluated on the Semantic Textual Similarity (STS) tasks yields an accuracy of 86.39%.

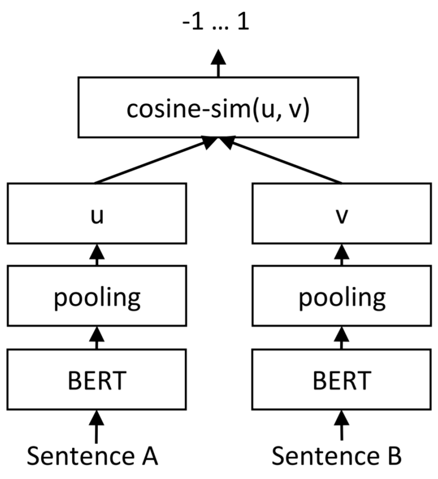


Figure 2.3: SBERT architecture (Reproduced from Figure 2 of [24]).

Pre-trained SBERT model shall be used for our case to extract semantic representations of users' textual data i.e., *my_story*. The choice of a pre-trained model is 'roberta-base-nli-stsb-mean-tokens' which describes that the pre-trained model is RoBERTa base with a mean pooling strategy and fine-tuned for STS (Semantic Textual Similarity) task. The base model provides a fixed vector size of 768. Table 2.2 shortlists all the user profile information along with their values, vector size and corresponding data type.

User feature	Value range	Size	Type of data
age	choose one from [18, 100]	1	numerical data
location	[latitude, longitude]	2	
i_am	choose three from [0, 18]	18	categorical data
meet_for	choose three from [0, 18]	18	
marital_status	choose three from [-1, 3]	5	
kids	choose three from [-1, 2]	4	
my_story	textual data	768	textual semantics

Table 2.2: Numerical, categorical and semantic data of a user profile.

2.4.7 Representation of user graph information

The goal here is to represent the user friend circle i.e., the local neighbourhood of the user in a mathematical form. The representation learning on graphs (Graph Neural Networks) have also undergone major developments in recent times. Some of the GNNs are capable of incorporating both user profile and graph information and achieve an encoder objective. The encoder objective, as previously described, is to encode the user characteristic such that the similarity or proximity in vector embedding space approximates friendship compatibility in the original network. In this section, we delve into various GNN-based encoders for this case.

The notion of neural networks for graph data was first outlined in Gori et al. (2005) [25] and further elaborated in Scarselli et al. (2009) [26]. Then, the graph embedding methods such as adjacency-based similarity (2013) [27], multi-hop similarity (2015) [28], random walk (2014) [29] and node2vec (2016) [20] showed improvements in learning graph representations. Node2vec has found to perform better on node classification while multi-hop methods for link prediction (2017) [30]. Node2vec has been fairly successful in generating friendship recommendations for a social network graph with heterogeneous edges (2019) [31]. However, these encoders have shortcomings such as linear parameter size, cannot incorporate node features, and inherently transductive i.e., it is not straight forward to generate embeddings for new nodes that were not included in the training set.

In contrast, deep encoders have shown its prowess in capturing the deep representations of graph structures. These encoders rely on the notion of 'Graph Convolutions Networks' (GCNs). This approach originated with the work of Bruna et al. (2013) [32], which presented a version of graph convolutions based on spectral graph theory. Following this work, GCNs have shown new state-of-the-art results on benchmarks such as node classification, link prediction, as well as recommendation tasks (e.g., the MovieLens benchmark [33]). Within GCNs, several variants are available such as GraphSAGE (2017) [34] and Gated Graph Neural Network (2016) [35].

A more recent application of Graph Convolution Networks (GCNs) named PinSage [1] has set a new benchmark in large-scale recommendation task. PinSage research work was collaborative research between Stanford and Pinterest. It addresses the large-scale item recommendations for the Pinterest platform.

PinSage proposes a random walk based GCN for web-scale recommender systems. Some of the features of PinSage include inductive learning i.e., the ability to generate embeddings for nodes that were not used in the training, the ability to accommodate node features, scalable to large graphs with billions of nodes. PinSage not only produces the graph representation but also achieves the encode objective of this case. All these merits make PinSage the right choice for this case. Thus, a modified variant of PinSage can be formulated for this case to generate rich embeddings for each user.

2.4.8 Friendship recommendations from node embeddings

The goal here is to use the node embeddings from the GCN to offer friendship recommendations. Intuitively, the nearest neighbour search can be made in the embedding vector space. Moreover, the PinSage mechanism employs the K-Nearest-Neighbor (KNN) technique on the node embeddings to offer recommendations showing excellent performance. Thus, even for our case, KNN search is employed. For each user, KNN search is performed on cosine distance metric and provides recommendation of K number of friends. Thereafter, the predicted recommendations are evaluated against the actual friendship formed to benchmark its performance.

2.5 Evaluation

Evaluation of the model shall be made wherein the time evolution of the data shall be used to define training, validation and test data. This is because to emulate the use case of the recommendation system, in which the current friendship connections are used to recommend persons who might become the user's friends in the future. More specifically, several snapshots of the state of the network shall be taken with a specific time interval between each snapshot.

The first snapshot will be used to define training dataset, the second snapshot to define validation dataset, and the third snapshot to define the test dataset. However, the user set is fixed from the training data and the new friend connections formed in the successive interval are accounted for evaluation. For instance, the model is trained using training snapshot data and made to predict its existing friendship connections and that gives the model performance on the training data. Suppose, the model is made to predict the new friendship connections, then it is evaluated against actual new friendship connections formed in the interval between training and validation snapshot data. This provides the model performance on the validation data. Similarly, the friendship predictions against actual new friendship connection between validation snapshot data and the test snapshot data provide the model performance on the test snapshot data.

After the development is complete, the performance benchmark on the test snapshot data provides the final performance of our model. Figure 2.4 shows how the evaluation is performed. We begin from a user and query for nearest neighbours on the vector embeddings space giving us top K prediction list. Then, this list is evaluated against actual friends that were formed in the period between the snapshots.

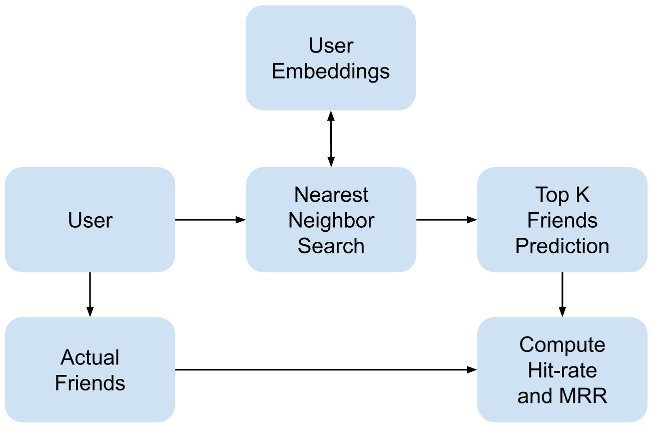


Figure 2.4: Evaluation of friend connection Predictions

To quantify the performance of recommendation model, we need to identify relevant evaluation metrics to optimise and to measure the model performance. Evaluation of a recommendation system can be divided into offline metrics and online metrics. Online metrics provide a measure of the effectiveness of the online recommendation system after the model roll-out into production[36]. Click-through rate and A/B testing are examples of online metrics. Offline metrics provide a measure of the effectiveness of the model during its development stage. Offline metrics are further categorised based on factors such as relevance and serendipity. Both of these metrics are covered separately in the following subsections.

2.5.1 Metrics of relevance

These metrics measure how accurate the predictions are when compared against actual friend connections that were formed in a specific interval of time. Some of the metrics of relevance are hit-rate and Mean Reciprocal Rank (Rank-aware).

Hit-rate (*relevant*)

Hit-rate is computed as the what fraction of the actual new friendship formed in an interval were present in the predicted top K recommendations. Hit-rate can take values between 0 to 100, where a value of 100 implies that all of the actual new friends are present in the top K predicted list. The value decays linearly with each missing actual friend in the prediction list. Subsequently, the average value of the hit-rate of all users is considered. K is chosen to be 500 for three reasons (1) it is a good start, (2) it provides sufficient user set for further stage i.e., user preference filter shown in 1.1 and (3) it is the number chosen in PinSage paper and gives a fair idea to compare.

Mean Reciprocal Rank (MRR) (*relevant*)

Mean Reciprocal Rank measures the ranking ability of a recommendation system and hence, falls also under the rank-aware metric. For a user, it is the reciprocal index of the first relevant match. The match is when an actual new friend formed is in the top K recommendations made by our model. This is applicable only if there is a match i.e., hit-rate is non-zero. MRR can take values between 0 to 100. Near to 100 imply that the actual new friends were very high up the list of predicted recommendations. The formula is shown in 2.1. $R_{u,f}$ is the rank of user f among recommended friends queried for user

u, U is the pool of all users in which it was queried and $|U|$ is the cardinality of set U [1]. Unlike hit-rate, MRR decays in a multiplicative inverse with each drop in matched rank. However, [1] uses a scaling factor of 100 and the reason to ensures that the difference between rank at 1,000 and rank at 2,000 is still noticeable, instead of being very close to 0.

$$MRR = \frac{1}{|U|} \sum_{(u,f)\in U} \frac{1}{R_{u,f}} \tag{2.1}$$

2.5.2 Metrics of serendipity

Metrics of serendipity indicates how interesting a recommendation list is. Even if the metrics of relevance is good, we may face the problems such as recommending the same set of friends to all users or the recommendation list is homogeneous (not diversified) or that it shows the same list for the users every single time [37]. [38] illustrates how focusing on accuracy alone can be detrimental to a recommender system. Example metrics are diversity, novelty, and unexpectedness. Diversity metric measures the variety in a recommendation list. Novelty metric measures how personalized is the recommendations i.e., how different is the list for different users. Unexpectedness measures how different the recommendation is from the previous recommendation for the same user.

However, for simplicity, we focus on offline metrics and mainly the metrics-of-relevance (accuracy metrics) for our case. The reason is accuracy metrics are foremost essential to understand the prediction accuracy. And our choices are hit-rate and mean reciprocal rank. Because these are the accuracy metrics that were considered in the research paper PinSage [1] on which our recommendation system is based. Further, hit-rate gives the picture about how accurate the predictions are while MRR gives the ranking ability of the prediction. Both of these together give a fair idea about the ability of our recommendation model.

2.5.3 Embedding similarity distribution

Another indication of the effectiveness of the learned embeddings is that the distance between random pairs of user embeddings are widely distributed [1]. If all the users are at about the same distance (i.e., the embeddings are tightly clustered) then the embedding space does not have enough “resolution” to distinguish between distinct users. Embedding similarity distribution gives a plot of cosine similarity between random pairs of user embeddings.

Another indicator of the effectiveness of learned embeddings is ‘kurtosis’. Kurtosis is a measure of "fatness" of distribution tail. Excess kurtosis indicates how fat the distribution is when compared against unit normal distribution. Figure 2.5 shows the values of excess kurtosis for various curves. Excess kurtosis value is 0 for a unit normal distribution. The steeper the curve is, the higher the excess kurtosis value gets.

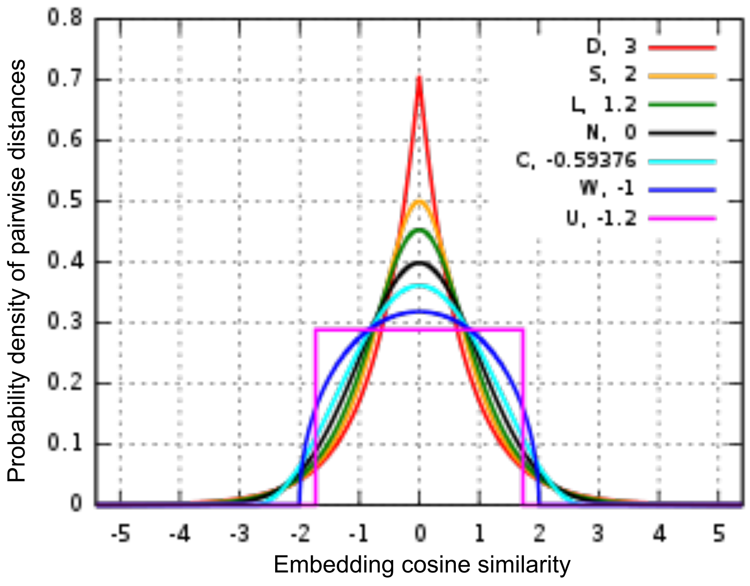


Figure 2.5: Distribution plot with excess kurtosis value. (Reproduced from [39].)

Chapter 3

Methods

This chapter illustrates several methods in detail that are chosen to solve this problem.

3.1 Data pipeline

In computing, a pipeline, also known as a data pipeline, is a set of data processing elements connected in series. Figure 3.1 shows the proposed data pipeline for the recommendation system. In our case, it involves stages: data collection, data preparation, model training, recommendations and lastly, the feedback to the model training stage after model evaluation. In fact, this pipeline can operate seamlessly from incorporating new user into the system to producing recommendations and model evaluation. Further, the periodical model evaluation serves as a decision for model training. That means suppose the performance is not satisfactory, we may fine-tune the model on-demand. Each stage of the data pipeline is explained in the following section.

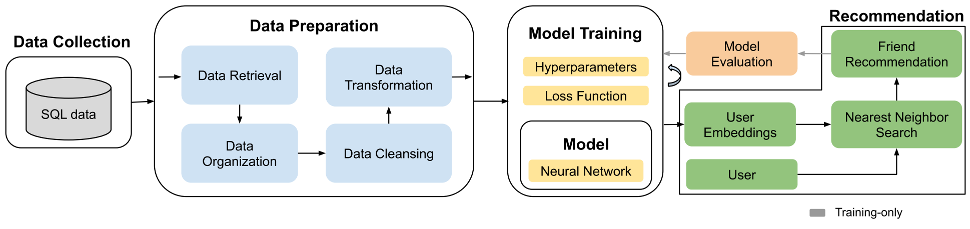


Figure 3.1: Data pipeline of friendship recommendation system

3.1.1 Data collection

As discussed in the evaluation section (background chapter) 2.5, we consider three time-series snapshot of the state of the social network from the SQL database, with specific time interval between each snapshot. The first snapshot is taken on 07^{th} of April 2020 and shall be used to define training dataset. The second snapshot is taken a month after on 07^{th} of May 2020 and shall be used to define validation dataset. Lastly, the third snapshot is taken three months after on 07^{th} August 2020 and shall be used to define test dataset.

3.1.2 Data preparation

Data retrieval

Data retrieval means obtaining relevant data from a database management system. Once we get the SQL dump, we set up a local SQL server and perform a relevant query using SQL scripts. The query is to extract user profile information such as *my_story, i_am, meet_for, marital_status, kids, birthday* and *location*. The query also involves gathering a list of friends and blocked user pairs on the platform. For the retrieval, we use multiple softwares such as 'SQL workbench' tool to set up a local SQL server, 'DBeaver' as a SQL client software to interact with the data, and 'pymysql' library to import data into our development environment as 'Pandas' dataframe and save the files in the HDF5 (Hierarchical Data Format) for further access.

Data organize

Data organizing is the process of preparing data for analysis by rearranging, segregating, aggregating, ordering, or modifying the structure of data that is cluttered or disintegrated. Raw data may hold data in multiple columns or may have them in randomly ordered, etc. Once the raw data has been obtained, we may need to organize the data. This includes subtasks such as segregation, aggregation, ordering and formatting. In the end, we obtain two sets of data. One is the organised user profile information, and the two is the list of friends and blocked user pairs.

Data cleansing

Data cleaning is the process of preparing data for analysis by filtering or modifying data that is incorrect, incomplete, irrelevant, duplicated, or improperly formatted. These type of data are not helpful when it comes to analyzing them and it may hinder the process or provide inaccurate results. We carry-out operations such as imputation i.e., assign default values to missing or erroneous values, removal of duplicates, emoticons and multiple punctuation marks from the textual data.

Data transformation

Data transformation is the process of converting a raw data source into a transformed, validated, and ready-to-use form. We transform numerical, categorical and semantics data into mathematical tensor. The categorical data into one-hot encoded vector while the numerical data is kept as it is. More importantly, the user profile text or *my_story* is translated to english language. Then, we use the pre-trained SBERT model to generate semantic embeddings. Google Cloud Platform (GCP) Translator API service was employed for the translation. However, due to the high-cost price of the GCP translator service, we consider only the Stockholm city users for our experiments that include user semantics and the number of users of Stockholm is 14,948.

3.2 Non-learned recommender system

The current friendship recommendation system at goFrendly is a non-learned system which was developed based on intuition. A non-learned recommender system in this case shortlists similar users based on profile information. This is implemented in our experiment the K nearest neighbour search is performed directly on the tensor which represents user profile information.

3.3 GCN-based encoder

3.3.1 Graph Convolution Network (GCN)

Graph convolution networks involve the generation of user embeddings for a target node from its node information and graph structure information. GCNs are known to produce rich embeddings compared to other graph embedding methods. Each successive degree connections of the node become a sequentially connected layer in its computational graph. The information flow begins from the highest degree connection nodes and undergoes neural operations getting transformed and aggregated across several layers (deep layers). The information flows from K-degree connections towards the target node thereby capturing the deep representations of the node's neighbourhood.

Figure 3.2 demonstrates the idea of GCN. At first computational graph is produced for each node and then, the GCN algorithm is applied. On the left side, a sample input graph is considered wherein neighbourhood aggregation is performed for the target user node A. The computational graph is produced wherein each successive layer from node A represents its n^{th} degree connections as shown on the right diagram in 3.2. Here, Layer-2 represents the target user node, Layer-1 is its first degree connections, and Layer-0 is its second-degree connections. The layers are named in the reverse order for the mathematical convenience. The node information in the Layer-0 gets transformed, aggregated and propagates across layers and forms the embedding of the target user node A. In general, Layer N represents the target node while Layer (N-1) represents its first degree connections, and Layer-0 its n^{th} degree connections. However, the [1] proposes random walk based computational graph and a novel GCN algorithm both of which shall be discussed in the next subsection.

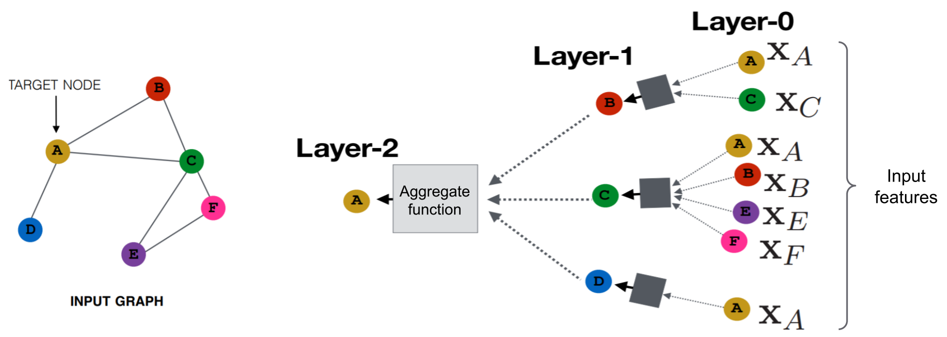


Figure 3.2: (1) A sample input graph with a target node A (left diagram), (2) Demonstration of GCN on node A (right diagram) (Adapted from Figure 1 of [1]).

3.3.2 PinSage

PinSage mechanism provides the techniques involving computational graph formation and aggregate function for our GCN algorithm [1]. Key techniques proposed in PinSage that shall be used to solve our friendship recommendation problem:

1. Random walk based computation graph for each node.
2. On-the-fly convolutions.
3. Importance pooling of neighborhood.

Figure 3.3 shows the overview of our neural network system. We begin from the user/node information and friends list, then create a social network graph. Thereafter, random walk is carried out for each node upon which K-layers computation graph is formulated. Starting from user features the PinSage convolution is applied onto each node's computation graphs yielding us with node embeddings. The operations such as random walk and PinConv shall be specifically explained in the next section.

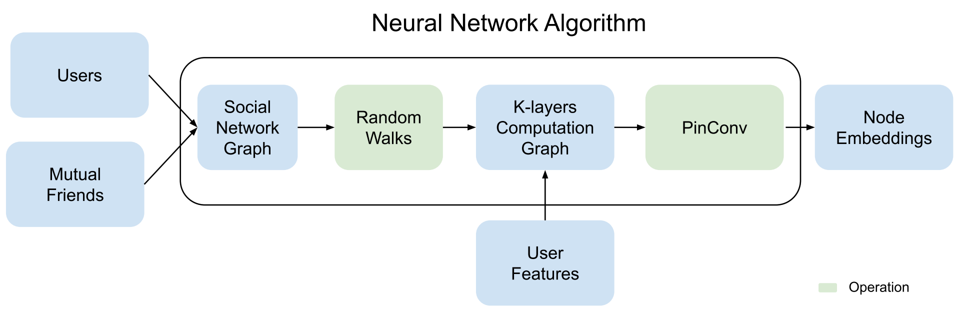
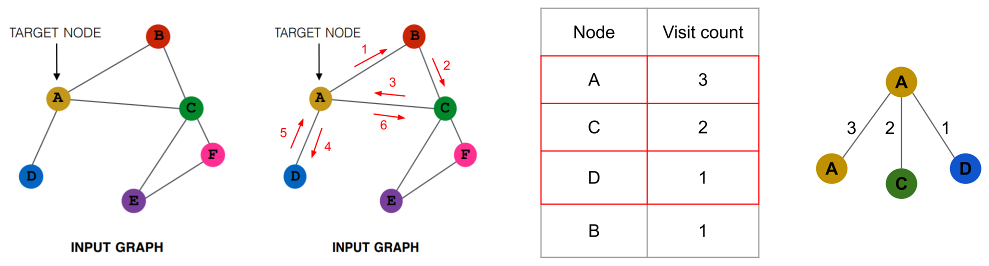


Figure 3.3: Neural network model involving PinSage based GCN.

Random walk

Computational graph for our convolution algorithm is based on the random walk method. The random walk gives the importance-based local neighbourhood, where the neighbourhood of node u is defined as the T nodes that exert the most influence on node u [1]. The random walk is simulated starting from each node and compute the visit count of nodes tread by the random walk. Random walk parameters include the number of walks, length of a walk, and the number of the top-visited nodes chosen (or the number of neighbours or

T). These parameters are the hyperparameters of a random walk.



Node	Visit count
A	3
C	2
D	1
B	1

Figure 3.4: An example of Random Walk operation. (Adapted from Figure 1 of [1].)

An example of a random walk is shown in figure 3.4. A small input graph is considered in this example and random walk shall be performed for the target node A. Its parameter values are chosen as (1) length of walk = 6, (2) number of walk = 1, (3) number of neighbors = 3. Then, T = 3 most visited nodes are considered with its visit counts as the weights. The obtained computational graph along with its weights for our target node A is shown on the far right diagram. With longer walks, farther neighbourhood can be spanned while more number of walks lead to larger coverage of nearby nodes. However, a large number of neighbours would sometime mean more similar neighbour nodes for many nodes. Thus, the choice of random walk parameters is important which shall be selected using random search.

PinConv

PinConv, a variant of GCN, performs an efficient localized convolution by sampling the local neighbourhood of a node. Algorithm 1 below describes the mathematical details of the aggregate function proposed in PinSage [1]. We consider the task of generating an embedding, z_u for a node u, which depends on the user's features and the graph structure around this user node u. This procedure is detailed in Algorithm 1 below. The basic idea is that we neural transform the representations z_v, $\forall v \in N(u)$ where $N(u)$ represents u's neighbours, through a dense neural network and then apply an aggregator/pooling function (e.g., an element-wise mean or weighted sum, denoted as γ) on the resulting vectors (Line 1). This aggregation step provides a vector representation, n_u, of u's local neighborhood, $N(u)$. Then the aggregated neighbourhood vector n_u is concatenated with user u's features z_u and neural transform through another dense neural network layer (Line 2). Additionally, the normalization in Line 3 makes the training more stable, and it is more efficient to perform an approximate nearest neighbour search for normalized embeddings. Q, q, W and w are the learnable parameters. The algorithm is shown below and is reproduced from [1].

```
Algorithm 1: CONVOLVE
Input  : Current embedding $\mathbf{z}_u$ for node $u$; set of neighbor
         embeddings $\{\mathbf{z}_v | v \in \mathcal{N}(u)\}$, set of neighbor weights
         $\boldsymbol{\alpha}$; symmetric vector function $\gamma(\cdot)$
Output : New embedding $\mathbf{z}_u^{\text{NEW}}$ for node $u$
1 $\mathbf{n}_u \leftarrow \gamma\left(\{\text{ReLU}\left(\mathbf{Q}\mathbf{h}_v + \mathbf{q}\right) \mid v \in \mathcal{N}(u)\}, \boldsymbol{\alpha}\right)$;
2 $\mathbf{z}_u^{\text{NEW}} \leftarrow \text{ReLU}\left(\mathbf{W} \cdot \text{CONCAT}(\mathbf{z}_u, \mathbf{n}_u) + \mathbf{w}\right)$;
3 $\mathbf{z}_u^{\text{NEW}} \leftarrow \mathbf{z}_u^{\text{NEW}} / \|\mathbf{z}_u^{\text{NEW}}\|_2$
```

3.4 Model training

Figure 3.5 shows the model training. User features and friends undergo the neural operation as described previously in figure 3.3. Then, the model produces user embeddings for which loss value is computed using the friend connections and blocked user pairs. Later, we carry-out backpropagation algorithm to update the model parameters.

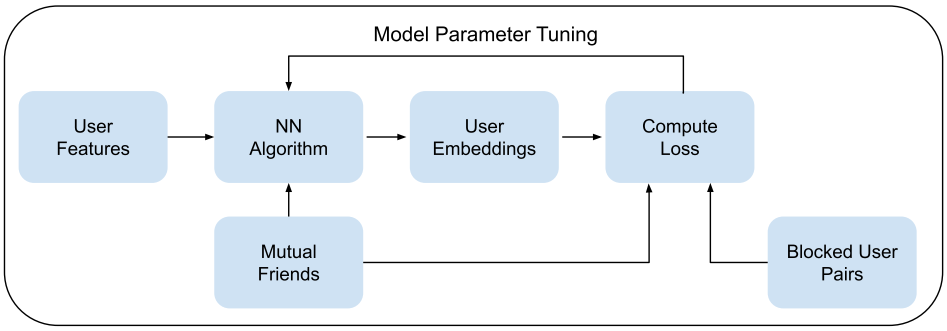


Figure 3.5: Model Training

3.4.1 Loss function

The loss function for the training of the neural network model is cosine embedding loss. Cosine embedding loss optimises the neural network to generate embeddings such that the embeddings of the friends are brought closer while the blocked users are pushed apart. The mathematical formula of cosine embedding loss function is shown in equation 3.1. Suppose x_1 and x_2 are the user pair while y's value represents whether they are friends or blocked users. Margin or also called as loss margin indicate the magnitude of the push for the negative user pairs.

$$Loss(x, y) = \begin{cases} 1 - cos(x_1, x_2), & \text{if } y = 1 \\ max(0, cos(x_1, x_2) - margin), & \text{if } y = -1 \end{cases} \tag{3.1}$$

3.4.2 Hyperparameters

In this entire setup, we have several hyperparameters, which we shall divide further based on operations as follows.

Hyperparameters 01 (Model configurations)

These hyperparameters are related to the blocks in neural network algorithm as shown in 3.3 and includes parameters related to random walk operation and the neural network. The hyperparameters of the random walk are the number of walks, length of walk and number of neighbours for each node. The hyperparameters of the neural network are the dimensions of convolutional layers, number of hidden layers and size of output node embeddings.

Hyperparameters 02 (Model training)

These parameters are related to model training. The model training hyperparameters include parameters such as loss margin, learning optimizer and learning rate.

3.4.3 Hyperparameter Tuning

In order to identify optimal model configurations, we carried-out hyperparameter tuning. Figure 3.6 shows the range of values that were tried to find the optimal values. The aim was to find the best value for hit-rate. The range is denoted with square brackets wherein the choice was a randomly picked value in the given range. Tuning was carried-out one variable at a time. The figure also shows the chosen values that performed the best.

Parameter	Range of values	Chosen
Length of walk	[8, 64]	64
Number of walks	[8, 64]	32
Number of neighbors	[4, 64]	16
Learning rate	[1e-5, 1e-2]	3e-4
Optimizer	‘SGD’, ‘Adadelta’, ‘Adagrad’, ‘Adam’, RMSprop’	‘RMSprop’
Loss margin	[-1,1]	0.25
Dim. of Convolution layer	[Feature dim / 5, Feature dim * 2]	Feature dim
Num. of hidden layers	[0, 4]	0
User embeddings size	[Feature dim / 6, Feature dim * 4]	Feature dim
Training samples (social connections)	Activity & mutual friends, blocked & viewed but not friends	Mutual friends & blocked pairs

Figure 3.6: Hyperparameters with its experiment values and its chosen value.

Chapter 4

Results and analysis

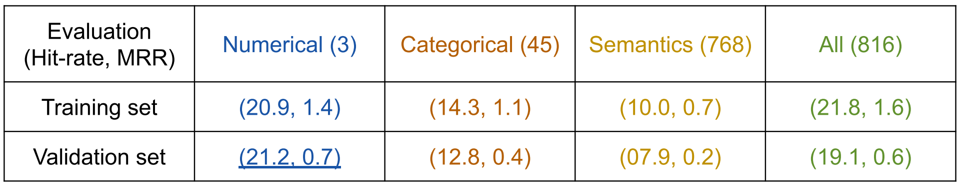
In this section, we shall benchmark the performance of three different recommendation models. The models are random recommender system, non-learned recommender system and GCN recommender system. Thereafter, we benchmark the performance of GCN recommender system for three different setups. All performance comparisons are made on validation dataset.

4.1 A random recommender system

At first, we evaluate the random recommender system which neither uses user profile information nor its friend circle information. Hence, user embeddings are uniformly randomly initialized. A uniform random recommender would mean selecting 500 users from 14,948 (Stockholm city) users at random that approximates to 3.33 in hit-rate and 0.66 in MRR.

4.2 Non-learned recommender system

As mentioned earlier, a non-learned recommender system enlists similar profile which is implemented by k nearest neighbours on the user input features. Figure 4.1 shows the non-learned recommender performance for numerical, categorical and semantic data individually and all in combination.



Evaluation (Hit-rate, MRR)	Numerical (3)	Categorical (45)	Semantics (768)	All (816)
Training set	(20.9, 1.4)	(14.3, 1.1)	(10.0, 0.7)	(21.8, 1.6)
Validation set	(21.2, 0.7)	(12.8, 0.4)	(07.9, 0.2)	(19.1, 0.6)

Figure 4.1: Non-learned recommender system performance.

From the above results, we may notice that each data class has shown different performances. Non-learned model is well generalized as the performance on training and validation data is comparable for all data classes. However, numerical data tops in performance by a significant value compared to others. This shows that numerical data have a direct influence on the friend connection formed. In other words, the users who are friends are closer to ‘age’ and ‘location’ values than other features.

Further, to get an idea about the distribution of these features we shall plot the embedding similarity distribution individually for each class and in a combination of all class. Figure 4.2 shows the plot. Further, excess kurtosis values for each have been computed and are 10.77, 1.03, -1.24, 4.44 for numerical, categorical, textual data respectively. Numerical data are clustered close to 1 with kurtosis indicating highest steepness whereas semantics data less clustered with least kurtosis value. In general, wider curves with lesser kurtosis are preferred as they are known to have better resolution. However, our results indicate otherwise i.e., higher kurtosis data has higher performance while lower kurtosis data has lower performance.

More specifically, the current recommendation system at GoFrendly combines numerical(3) and categorical(45) data. Numerical data consists of *age* and *location* whereas categorical data consists of *i_am, meet_for, marital_status, kids*. Thus, we shall benchmark the results for these two cases and later extend to include semantics data. As we have the semantics data of Stockholm city users alone, our user size here is 14,948. Figure 4.3 shows the performance of random recommender as well as the non-learned system for the same.

It is evident from the table that the non-learned recommender system performs significantly better. It is worthwhile to note that the model has excellent generalization i.e., the performance on the validation data was compared with the performance on the training data. The loss value was found to be 0.089.

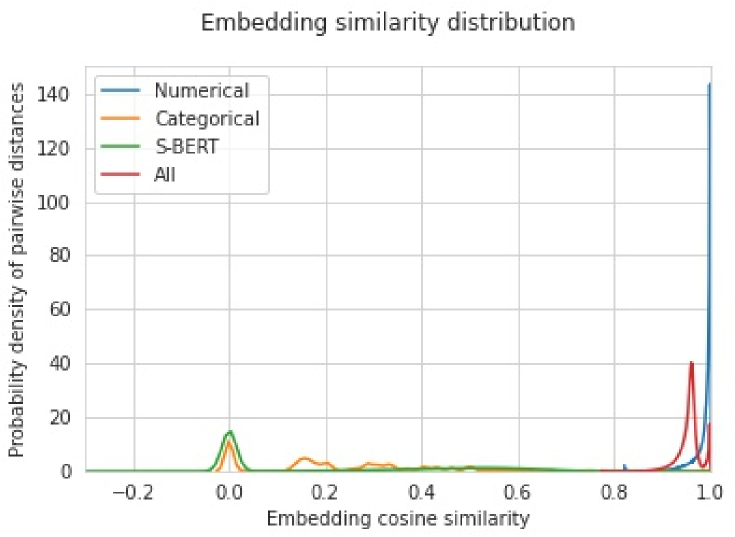


Figure 4.2: Embedding similarity distribution of the user features

Evaluation (Hit-rate, MRR)	Numerical & Categorical data of Stockholm city users (48 features)	
	Random model	Non-learned model
Train data	(3.4, 0.3)	(28.2, 2.9)
Validation data	(3.9, 0.1)	(26.4, 1.3)

Figure 4.3: Comparison of performances of random and non-learned models

In the next section, we shall develop a deep GCN based encoder to generate rich user embeddings and compare the performances.

4.3 GCN-based encoder

Our PinSage based GCN model underwent training with node information as initial node embeddings. The node information includes numerical and categorical data accounting to 48 and the user set was limited to Stockholm city i.e., 14,948. Figure 3.1 shows the data pipeline and describes how the training is carried out. The hyperparameter values are chosen from its tuning as presented in figure 3.6.

Figure 4.4 shows the loss values over the course of training. The green line overlay represents the loss value of the non-learned system. It is noticeable that the value reduced smoothing affirming the choice of our training optimizer and learning rate. The training consisted of 9,745 trainable model parameters.

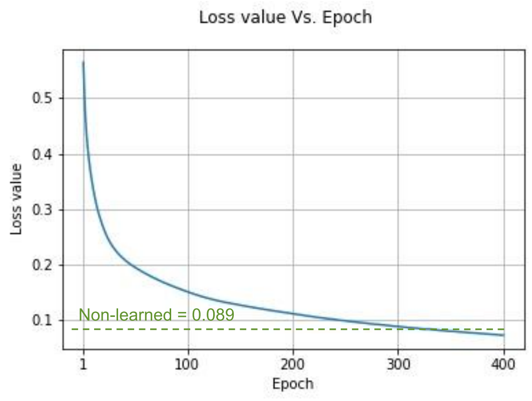


Figure 4.4: Loss values during model training

Figure 4.5 shows the model's hit-rate and MRR with respect to training epochs. It is worth noticing that the hit-rate and MRR have gradually improved over the course of training and reached a new benchmark. However, the performance gap between training and validation indicate overfitting. Generalization techniques such as including the layers of dropout, reducing the number of hidden layers, and early stopping were implemented. All these attempts had reduced the training-validation curve gap and the best is presented here in figure 4.5. However, further generalization could not be achieved.

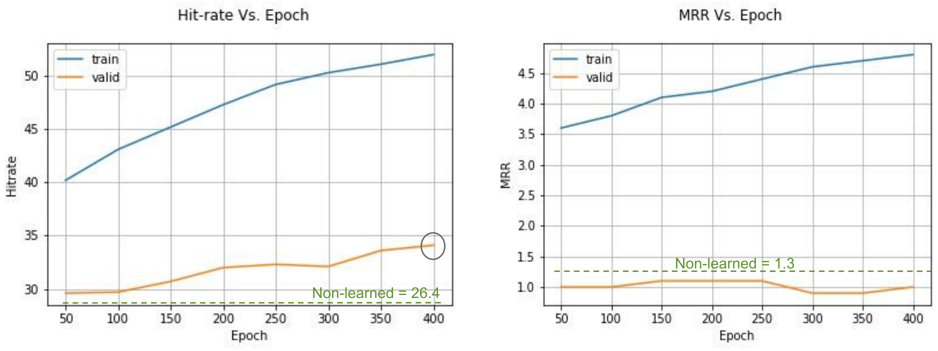


Figure 4.5: Hit-rate and Mean Reciprocal Rank (MRR) during training.

The green line overlay represents the performance of the non-learned system on the validation data and a baseline for our comparison. The best Hit-rate (marked with a black circle) were seen to be 52.0 and 34.1 on training and validation data respectively. Both of these values surpass the non-learned recommender system (performance on validation data) by a huge margin. Furthermore, the top value of MRR was 4.8 and 1.0 on training and validation data respectively. Although MRR has risen for the training data, it has shown no improvement for the validation data compared against non-learned MRR. Thus, the model training needs to terminate at the point corresponding to the maximum hit-rate performance on validation data and a baseline performance for further comparisons.

Moreover, we shall plot similarity distribution of the embeddings taken at regular epoch intervals. This allows us to understand the evolution of the distribution of user embeddings at training intervals. Figure 4.6 depicts the embedding similarity distribution during training intervals. We observe top performance at epoch 400. Although the curves have a similar distribution, we may notice that the embeddings corresponding to epoch 400 are more clustered (less wide) than others. Besides, excess kurtosis was computed and were -1.44, -1.45, -1.51, -1.49 for epoch 100, 200, 300 and 400 respectively. Excess kurtosis values are almost the same to compare. All these observations coupled with our previous observation in figure 4.2 indicate that embedding similarity distribution and kurtosis values are not positively correlated with the model's performance.

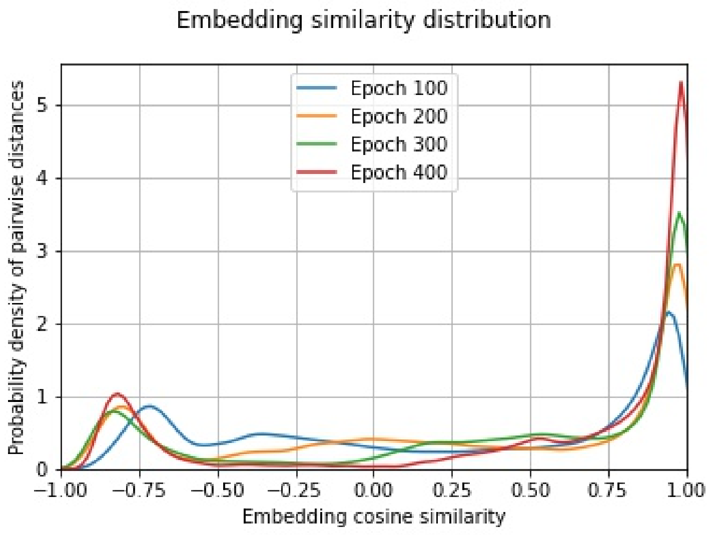
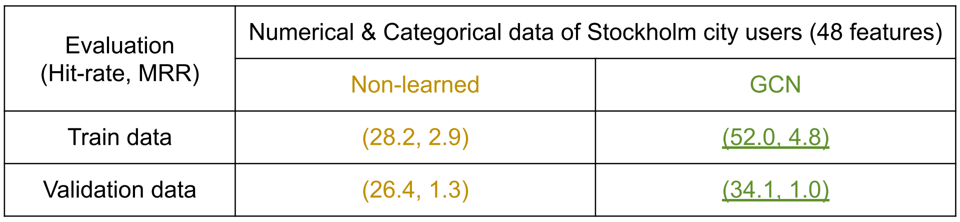


Figure 4.6: Similarity distribution of the GCN produced user embeddings

4.4 Models comparison

Figure 4.7 compares the performance between non-learned and GCN-based recommender system. It is evident that GCN architecture allowed the model to capture the friendship patterns well leading to almost twice as non-learned model's performance on both hit-rate and MRR for training data. Furthermore, the hit-rate improvement on validation dataset is significant by 29.16% over non-learned. However, no significant improvement in MRR for the validation data was noticed.



Evaluation (Hit-rate, MRR)	Numerical & Categorical data of Stockholm city users (48 features)	
	Non-learned	GCN
Train data	(28.2, 2.9)	(52.0, 4.8)
Validation data	(26.4, 1.3)	(34.1, 1.0)

Figure 4.7: Comparison between non-learned model and GCN model

4.5 Experiments

In order to improve the performance, for the same GCN design, we shall carry-out experiments such as (1) include user semantics information and (2) increase the training user-set to users of all cities.

4.5.1 Include user semantics information

In this case, we extend the user features to include numerical, categorical, and semantics data with a data size of 3, 45, and 768 respectively. User features now consist of *my_story, i_am, meet_for, marital status, kids, age* and *location* leading to a total of 816 vector size. However, the user size is fixed to Stockholm city users.

The training consisted of 2,672,401 trainable model parameters for this case. The choice of hyperparameters was the same except that the learning rate was 1e-5. Figure 4.8 shows the loss values during the training. The green line overlay indicates the loss value of our best performing GCN model so far (presented in section 4.3). Once again, the training was smooth and credits our choice of learning rate and optimizer. Figure 4.9 shows the hit-rate and

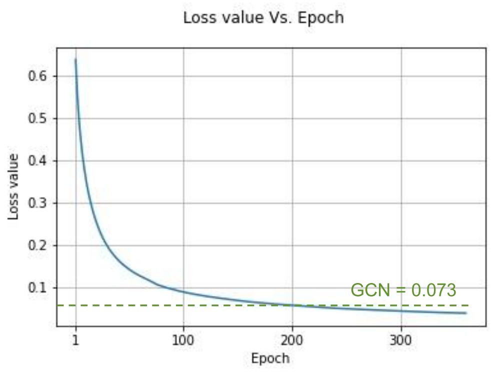


Figure 4.8: Loss values

MRR during the training. Looking at both the plots, we can notice that the hit-rate and MRR have scaled new benchmarks. This proves that with rich user features, the model was better able to identify its pattern with friendship formation. However, the model has not performed very well on the generalization as we see a big gap between these curves. Although generalization techniques have already been applied, no further improvements could be made.

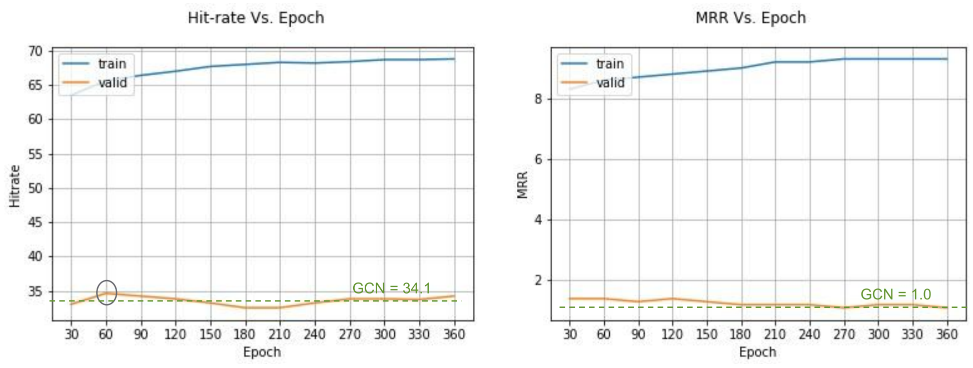


Figure 4.9: Hit-rate and Mean Reciprocal Rank (MRR) during training.

Furthermore, the hit-rate seems to oscillate around 34 whereas MRR has stayed near to 1 over the course of training. A well trained-model is one which performs the best on the validation dataset. Hence, the choice of node embeddings is training epoch 60 (marked with a blacked circle) with Hit-rate = 65.6, MRR = 8.6 on the training dataset, and Hit-rate = 34.6, MRR = 1.4 on the validation dataset. Both of these results outperform the previous GCN setup.

4.5.2 Include all users of the platform in the training

In this case, we extend the training userset to include users of all cities. The user features now include numerical and categorical data with a vector size of 3 and 45 respectively. User features now consist of ~~*my_story*~~, *i_am, meet_for, marital status, kids, age* and *location*. Our evaluation remains to be on the Stockholm city users.

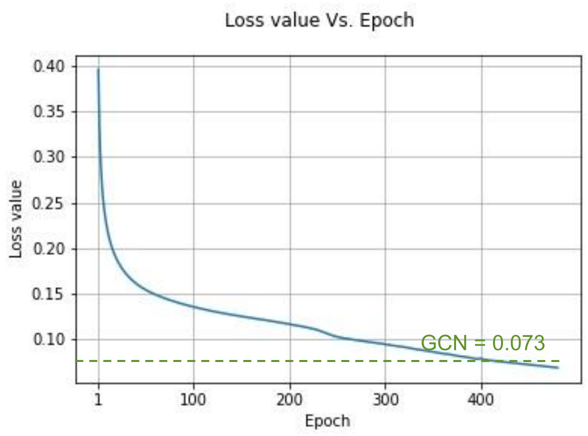


Figure 4.10: Loss values during training

The training consisted of 9,745 trainable model parameters for this case. The choice of hyperparameters was the same except that the learning rate was 4e-4. Figure 4.10 shows the loss values during the training. The green line overlay indicates the loss value of our first GCN model (presented in section 4.3) at its optimal performance. Once again, the training was smooth and credits our choice of learning rate and optimizer. Figure 4.11 shows the hit-rate and MRR during the training. Looking at both the plots, we can notice that the hit-rate and MRR have not performed better than our previous GCN benchmark. Although the hit-rate has improved over the course of training, its best value (marked with a blacked circle) on validation dataset is much lower than the previous GCN's benchmark. Further, MRR, yet again, has remained almost the same. However, we observe that this model generalizes better than the previous one as the gap between training and validation performance is seen to be smaller.

A well trained-model is one which performs the best on the validation dataset. Since the MRR has remained almost the same. The best result is Hit-rate = 34.8, MRR = 2.6 on training dataset, and Hit-rate = 29.2, MRR =

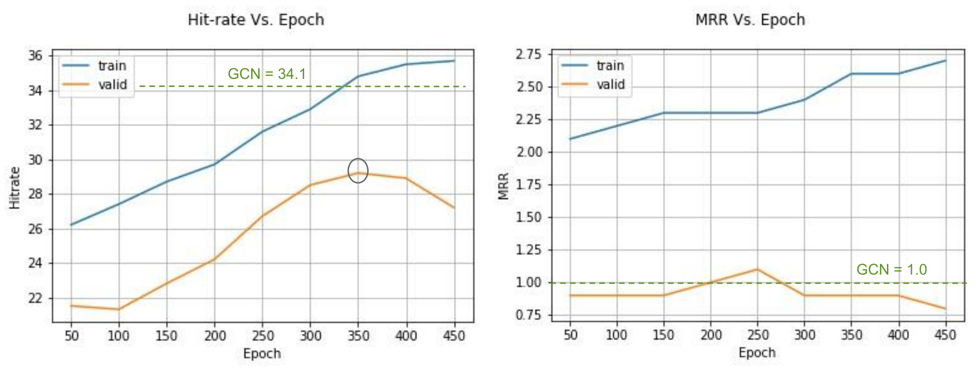
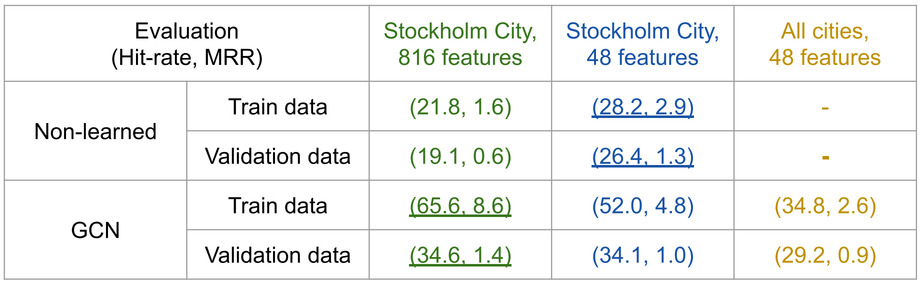


Figure 4.11: Hit-rate and Mean Reciprocal Rank (MRR) during training.

0.9 on validation dataset. Both of these results underperform when compared with our previous GCN setup. On the whole, training the model individually for a particular city yields better hit-rate and MRR than to train on all cities.

4.6 Models comparison

So far, we have had two different models viz. non-learned recommendation system and GCN-based recommendation system. And we had three different setups (1) limited user features trained on Stockholm city users, (2) rich user features trained on Stockholm city users, and (3) limited user features trained on users of all cities. Figure 4.12 shows the performance of both these models on three different setups. Top-performing cases are underlined for visual convenience.



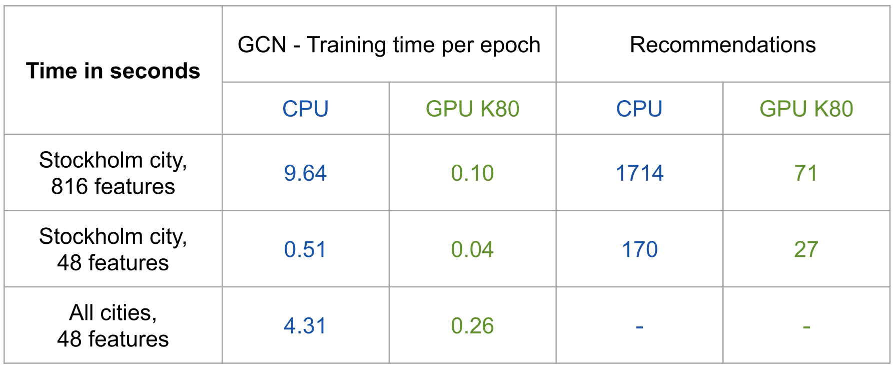
Evaluation (Hit-rate, MRR)		Stockholm City, 816 features	Stockholm City, 48 features	All cities, 48 features
Non-learned	Train data	(21.8, 1.6)	(28.2, 2.9)	-
	Validation data	(19.1, 0.6)	(26.4, 1.3)	-
GCN	Train data	(65.6, 8.6)	(52.0, 4.8)	(34.8, 2.6)
	Validation data	(34.6, 1.4)	(34.1, 1.0)	(29.2, 0.9)

Figure 4.12: Comparison between non-learned model and GCN model for three different cases.

From these results, we may deduce that a non-learned model performs best on limited user features for a chosen city, which is the current recommendation system at GoFrendly. However, GCN-based recommendation system significantly outperforms the non-learned system for the same case. For this GCN model, the setup with rich user features trained on a single city yields the top performance. Thus, we may conclude that a GCN-based model performs better than the non-learned model. Intuitively if the evaluation is made by offering to all users of all cities, the model will have a large pool of users to choose from which reduces its performance. This was also seen to be the case when evaluated. Furthermore, we can also deduce that individual encoder for each city can be developed for optimal performance. Lastly, we observe a significant performance gap between training and validation dataset indicating scope for model's generalization.

4.7 Timing profile

In all the aforementioned setups training and evaluation took a notable amount of time and figure 4.13 shows the timing profile.



Time in seconds	GCN - Training time per epoch		Recommendations	
	CPU	GPU K80	CPU	GPU K80
Stockholm city, 816 features	9.64	0.10	1714	71
Stockholm city, 48 features	0.51	0.04	170	27
All cities, 48 features	4.31	0.26	-	-

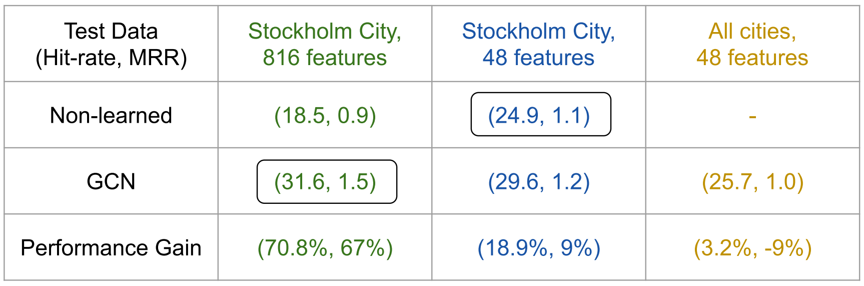
Figure 4.13: Timing profile of GCN model (in seconds).

From the above figure, we can notice that the time taken by the CPU is multi-fold when compared to GPU. Training time would translate to large time when trained for 100s of epochs. In the case of CPU, we observe that the increase in feature size by a factor of 17 increased the training time by close to 17 times. Similarly is the case for the large training user size. Time taken for the recommendations reflected a factor of 10 multi-fold.

However, to speed-up, the tensors were pushed to GPU to compute & relevant algorithmic functions were modified to GPU compatible. Hence, significant optimization in time was observed with GPU. In the case of GPU, we observe that the increase in feature size led to a slight jump in training time whereas the training time for the larger user case was a significant increase. The shown GPU timing is for Nvidia K80; however, when we considered Nvidia P100 GPU, the time reduced further by half. This demonstrates the merits of using GPUs for neural networks. Due to the availability of free GPU resources[1] and its computational merits, GPU has been the first choice for training neural networks.

4.8 Evaluation on the test data

Now, we consider the test data and evaluate our model performance. Figure 4.14 enlists the values for non-learned and GCN models for three different cases. The underlined values are top-performing cases. The results on test data are similar to the results on validation data i.e., GCN model for Stockholm city outperforms non-learned system on both hit-rate and MRR. Performance gain between non-learned and GCN model is shown below.



Test Data (Hit-rate, MRR)	Stockholm City, 816 features	Stockholm City, 48 features	All cities, 48 features
Non-learned	(18.5, 0.9)	$\boxed{(24.9, 1.1)}$	-
GCN	$\boxed{(31.6, 1.5)}$	(29.6, 1.2)	(25.7, 1.0)
Performance Gain	(70.8%, 67%)	(18.9%, 9%)	(3.2%, -9%)

Figure 4.14: Performance gain between non-learned and GCN model on test data.

The top performing cases are highlighted with black box in the figure above. The GCN model with full feature size trained on Stockholm city users tops the performance on hit-rate and MRR at 70.8% and 67% respectively. GCN model with limited feature size for the Stockholm city yields a moderate gain of 18.9% and 9% on Hit-rate and MRR. Lastly, the GCN model with limited features trained on all cities does not perform well enough and it proves that the GCN model for single city is a better design. Finally, the comparison

[1]https://colab.research.google.com/notebooks/intro.ipynb

of the current goFrendly recommendation system which uses limited features size against our top-performing GCN model shows that we achieve a gain of 26.9% on hit-rate and 36.4% on MRR (highlighted in black boxes). On the whole, our results demonstrate the merit of GCN model in recommending individual friendship connections.

Chapter 5

Discussion and Conclusions

This chapter presents a discussion and conclusions on the outcomes of our research work. It begins with a general discussion about the overall conduct of the thesis work. Then, the practical usefulness of the work along with societal, ethical and sustainable aspects in the real world are described. Thereafter, the conclusions that are based on the outcomes of this thesis work are presented. Lastly, we shall discuss the scope for further studies and research on this research problem.

5.1 General discussion

The thesis started with a clear research question 'can a suitable scalable deep neural network perform better than a non-learned friendship recommender for a given social network, and, if so, by how much?'. To test this hypothesis, we benchmark the performances of non-learned system and deep neural networks and then compare them on a common evaluation metric. Before the start of this project, the social network platform did not have pre-existing neural network compatible recommendation data pipeline in place.

In the literature study, we reviewed several methods on the lines of deep neural networks and recommendation systems. More emphasis was given for peer reviewed state-of-the-art methods to solve this case. Our study led to the choice of relevant and essential evaluation metrics. The non-learned recommender system was implemented and its performance was recorded. Our study also led to the shortlisting of using pre-trained SBERT language model and the PinSage graph network model considering their recent successes. PinSage is a paper from top conference KDD with over 500 citations at the time of writing this report and it is a modern high-performance graph convolution network applied for a large-scale recommendation.

We developed the recommendation pipeline with well-defined evaluation metrics. The proposed data pipeline is flexible to operate on-demand. Then, we trained our PinSage model iteratively by monitoring its prediction performance. The flexibility of our model allowed us to experiment with different setups leading to two more major experiments whose findings showed that a GCN model with full feature size trained for a single city gives optimal performance. Our results demonstrate the merit of GCN in offering friendship recommendations and speaks for itself when it comes to its future potential.

5.1.1 Push and pull problem

Another interesting observation during the model training was the phenomenon of push and pull problem. Our loss function was designed to pull the embeddings of friends (positive sample) closer and push the embeddings of blocked user pairs (negative samples) apart. However, if a user pulls two of her friends closer who are blocked users with each other, then the placement of these users in the embedding cannot be successful. This phenomenon could reduce the performance of the model for such cases. This, however, might be solved by having an individual model for each user with its own positive and negative samples. Although the individually trained model is computationally expensive, we only need to train once for each user and thereafter fine-tuning the model would suffice.

5.1.2 Non-GCN approach to solve the problem (MLP based encoder)

Since our PinSage based GCN model was technically complex to build and train, we developed a multi-layer perceptron based encoder as a start to study the compatibility of our recommendation data pipeline with the neural network models and also, to ensure smooth training for our chosen loss function. This model is capable of incorporating user features and 1^{st} degree connections and achieve the encoder objective. The model was a multi-layered perceptron with ReLU activation layers and batch normalization layer. The model was trained for the case of Stockholm city users with limited user features. After the hyperparameter tuning, we obtained a top performance of hit-rate of 27.4, MRR of 1.7 on training data, and hit-rate of 27.1, MRR of 0.8 on validation data. This performance is comparable with the non-learned system. Hence, we dismissed any further experiments on this model and considered PinSage based GCN for our further experiments.

5.2 Practical usefulness

Our results have shown good performance on the offline prediction accuracy metrics. Although accuracy metrics are foremost essential metrics for recommendation systems, it does not suffice to be called as an excellent recommendation system. Study shows that being accurate is not enough and using accuracy metrics alone may hurt the recommender systems [38]. Other metrics such as metrics of serendipity and online metrics are important to be in place. Our data pipeline and GCN model provides a reliable framework for further developments in the recommendation system.

The ranking ability of our GCN based recommendation model has remained a single-digit performance. If this is not satisfactory, we could use the GCN predictions as a user pool and rearrange them based on the heuristic ranking model, which is currently in use at goFrendly. Another point is that our recommender system is trained to capture the friendship patterns seen in a particular city. However, if an individual's friendship pattern differs from the city's crowd pattern, it could be a downside for such minority. In such cases, an individual friendship recommender trained exclusively on each user might solve this shortfall.

5.3 Societal, sustainable and ethical Implications

Going by the results, it would be interesting to witness the model deployed in the real world. With a better performance than the predecessor and the ability to iteratively learn, it is evident that such models shall be eventually deployed shortly. Also, this is a stepping stone for the developments of several other related features such as the recommendation of entities such as social activities.

Friendship recommendation systems are proven to boost user satisfaction leading to user retention and also known to provide user personalization which leads to users in a better mood to use the services [40]. Therefore, this research work directly addresses the business value of social network service providers and in turn, it addresses the economic aspects of our society. One of the core services of any social network platform is to facilitate the social dynamics of its user community and our friendship recommender system is one such facilitator. This feature specifically benefits users who particularly log into the platform to find friends and they shall find it with much ease than the non-learned way. On the other hand, the social experience following an online friendship is not assured to positive.

For the recommendations to thrive and receive appreciations from its users, it has to ensure the lists are interesting. This leads us to focus on metrics of serendipity and regular fine-tuning of the models. With a fair value of diversity, novelty and unexpectedness coupled with regular fine-tuning, the model shall be able to sustain the interest of its users longer. In contrast, rendering such elegant and accurate predictions would translate to a lot of cost and computing resources. The computing resources contribute to global CO2 emissions. However, the growing interest among the cloud computing service providers to reduce carbon footprint by methods such as usage of renewable energy takes us a step closer to a sustainable usage.

Users need to allow their personal information to be used to offer them valuable services such as friendship recommendations in return. Without such useful information, the model gives poor results as seen in random recommender system. The current non-learned system at goFrendly uses profile information to offer friendship recommendations. Our model, in addition, uses the friend circle information of each user and hence, only this information is to be notified to the user for approval. It is important that the privacy of all the users are to be protected. On the ethical aspect, this research work was carried out with regard to user privacy protection. Privacy is defined as the right to have some control over how one's personal information is collected and used[1]. In this regard, the users need to be informed that their personal information shall be used regularly to better the service (to fine-tune our models) offered to them and is protected.

[1]https://iapp.org/about/what-is-privacy

5.4 Conclusions

This thesis was an effort towards research and development of friendship recommendation engine for the social network ‘goFrendly’. In particular, to identify and apply state-of-the-art methods available in the field of recommendation systems. The data considered was a snapshot of the social network. The data had to be organized, cleaned and transformed. Our prestudy involved review of the recent technological advancements in this field and our influenced our choice of methods. Our selected methods were able to consider both user profile information as well as user friends circle up to K-degrees to offer recommendations.

The proposed mechanism involved an end-to-end data pipeline equipped with SOTA SBERT model for semantic extraction and PinSage model to elicit friend connection patterns. SBERT is a pre-trained language model to extract semantics from user textual data leading to rich user features. PinSage is the Graph Convolution Network that has been utilized in the thesis, along with pre-trained SBERT for semantic extraction.

In our experiments, PinSage model trained for a single (Stockholm) city users with full feature size stood out to be a top performer with a gain of 26.9% on hit-rate, 36.4% on MRR against the current non-learned model, which has limited feature size. This demonstrates the potential of the pre-trained SBERT, the GCNs and the multitude of recommendation problems that it can be applied.

5.5 Scope for Future Work

Even after achieving such good performance, many future improvements are possible. Considering the knowledge and insights acquired over the past six months, there are some interesting possibilities which can further be investigated.

First and foremost would be to include online evaluation metrics such as click-through rate for the existing recommendation data-pipeline. This enables us to understand the impact of prediction accuracy on the real world and further iterate our algorithms. Secondly, further study into a generalization of GCN models would give a significant advantage in its performance. Thirdly, we observed that each city has different friend connection patterns and an encoder for individual city performs well. On this line, we can examine having a GCN encoder for each user and perhaps a couple with curriculum learning proposed in [1].

Fourthly, we need to include the metrics of serendipity and observe its impact on online metrics. Further, the speed of recommendation can be improved by using Locality Sensitive Hashing (LSH), a scalable solution to offer quick K-NN results. Finally, we may use TPU (Tensor Processing Units) in our development environment to train, evaluate and iterate at a faster pace. Lastly, more quality data can be used for model training such as chat friends i.e., users who have exchanged messages.

5.6 Summary

The business problem was to develop a modern friendship recommendation feature for the given social network. Then the research question was formulated as 'Can a suitable scalable deep neural network perform better than a non-learned friendship recommender for a given social network, and, if so, by how much?'. The literature survey highlights a thorough survey on the recommender systems for graph-based recommendation problem and highlights the merits of 2018 KDD conference paper, namely, PinSage for solving a web-scale recommendation problem.

Particularly, the methods: random walk based computation graph, importance based pooling of neighbourhood and graph convolutional network algorithm were adapted to suit the case of the social network graph. These methods were employed to position friendship compatibility of each social network user in the mathematical space based on historical friendship links and then predict potential new friendship connections. The model was successfully trained for optimal recommender performance, namely, hitrate and mean reciprocal rank. The results show that the graph convolution network based model performance is notable and corroborates with its recent success. The analysis of the results indicate possible performance gain in building one model for each city. This project also provides numerous pointers for further research work in graph based recommender system.

Bibliography

[1] Rex Ying et al. “Graph Convolutional Neural Networks for Web-Scale Recommender Systems”. In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*. KDD ’18. London, United Kingdom: Association for Computing Machinery, 2018, pp. 974–983. ISBN: 9781450355520. DOI: 10.1145/3219819.3219890. URL: https://doi.org/10.1145/3219819.3219890.

[2] Dimitris V. Kalamaras. *‘Social Network Visualizer - Manual’*. https://socnetv.org/docs/index.html. [Online; last accessed 02-Apr-2020].

[3] Jonathan A. Obar and Steve Wildman. “Social media definition and the governance challenge: An introduction to the special issue”. In: *Telecommunications Policy* 39.9 (2015). SPECIAL ISSUE ON THE GOVERNANCE OF SOCIAL MEDIA, pp. 745–750. ISSN: 0308-5961. DOI: https://doi.org/10.1016/j.telpol.2015.07.014. URL: http://www.sciencedirect.com/science/article/pii/S0308596115001172.

[4] Statista. *Number of social media users worldwide from 2010 to 2021*. https://www.statista.com/statistics/278414/number-of-worldwide-social-network-users/. [Online; last accessed 17-Feb-2020]. 2019.

[5] Shuai Zhang et al. “Deep Learning Based Recommender System: A Survey and New Perspectives”. In: *ACM Comput. Surv.* 52.1 (Feb. 2019). ISSN: 0360-0300. DOI: 10.1145/3285029. URL: https://doi.org/10.1145/3285029.

[6] Carlos A. Gomez-Uribe and Neil Hunt. “The Netflix Recommender System: Algorithms, Business Value, and Innovation”. In: *ACM Trans.*

Manage. Inf. Syst. 6.4 (Dec. 2016). ISSN: 2158-656X. DOI: 10.1145/2843948. URL: https://doi.org/10.1145/2843948.

[7] James Davidson et al. "The YouTube Video Recommendation System". In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: Association for Computing Machinery, 2010, pp. 293–296. ISBN: 9781605589060. DOI: 10.1145/1864708.1864770. URL: https://doi.org/10.1145/1864708.1864770.

[8] Business Insider Inc. Nathan Mcalone. *'Why Netflix thinks its personalized recommendation engine is worth $1 billion per year'*. https://www.businessinsider.in/Why-Netflix-thinks-its-personalized-recommendation-engine-is-worth-1-billion-per-year/articleshow/52754724.cms. [Online;last accessed 20-May-2020]. 2016.

[9] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[10] Gerald Tesauro. "Temporal Difference Learning and TD-Gammon". In: *Commun. ACM* 38.3 (Mar. 1995), pp. 58–68. ISSN: 0001-0782. DOI: 10.1145/203330.203343. URL: https://doi.org/10.1145/203330.203343.

[11] Ian J. Goodfellow et al. "Generative Adversarial Nets". In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'14. Montreal, Canada: MIT Press, 2014, pp. 2672–2680.

[12] Jure Leskovec and Rok Sosič. "SNAP: A General-Purpose Network Analysis and Graph-Mining Library". In: *ACM Trans. Intell. Syst. Technol.* 8.1 (July 2016). ISSN: 2157-6904. DOI: 10.1145/2898361. URL: https://doi.org/10.1145/2898361.

[13] Linkedin Help Section Page. *'People You May Know Feature - Overview'*. https://www.linkedin.com/pulse/people-also-viewed-who-how-got-onto-my-profile-sid-clark/ and https://www.linkedin.com/help/linkedin/answer/29. [Online; last accessed 02-Apr-2020]. 2019.

[14] Facebook Help Section Page. *'People You May Know'*. `https://www.facebook.com/help/336320879782850/?helpref=hc_fnav`. [Online; last accessed 02-Apr-2020].

[15] William L. Hamilton, Rex Ying, and Jure Leskovec. "Representation Learning on Graphs: Methods and Applications". In: *CoRR* abs/1709.05584 (2017). arXiv: `1709.05584`. URL: `http://arxiv.org/abs/1709.05584`.

[16] Tomas Mikolov et al. "Distributed Representations of Words and Phrases and Their Compositionality". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Lake Tahoe, Nevada: Curran Associates Inc., 2013, pp. 3111–3119.

[17] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2013. URL: `http://arxiv.org/abs/1301.3781`.

[18] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. Ed. by Alessandro Moschitti, Bo Pang, and Walter Daelemans. ACL, 2014, pp. 1532–1543. DOI: `10.3115/v1/d14-1162`. URL: `https://doi.org/10.3115/v1/d14-1162`.

[19] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, Apr. 2017, pp. 427–431.

[20] Aditya Grover and Jure Leskovec. "Node2vec: Scalable Feature Learning for Networks". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 855–864. ISBN: 9781450342322. DOI: `10.1145/2939672.2939754`.

[21] Ashish Vaswani et al. “Attention is All You Need”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 6000–6010. ISBN: 9781510860964.

[22] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[23] Yinhan Liu et al. “RoBERTa: A Robustly Optimized BERT Pretraining Approach”. In: *CoRR* abs/1907.11692 (2019). arXiv: 1907.11692. URL: http://arxiv.org/abs/1907.11692.

[24] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”. In: *CoRR* abs/1908.10084 (2019). arXiv: 1908.10084. URL: http://arxiv.org/abs/1908.10084.

[25] M. Gori, G. Monfardini, and F. Scarselli. “A new model for learning in graph domains”. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2. 2005, 729–734 vol. 2. DOI: 10.1109/IJCNN.2005.1555942.

[26] Franco Scarselli et al. “The Graph Neural Network Model”. In: *Trans. Neur. Netw.* 20.1 (Jan. 2009), pp. 61–80. ISSN: 1045-9227. DOI: 10.1109/TNN.2008.2005605.

[27] Amr Ahmed et al. “Distributed Large-Scale Natural Graph Factorization”. In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW ’13. Rio de Janeiro, Brazil: Association for Computing Machinery, 2013, pp. 37–48. ISBN: 9781450320351. DOI: 10.1145/2488388.2488393.

[28] Shaosheng Cao, Wei Lu, and Qiongkai Xu. “GraRep: Learning Graph Representations with Global Structural Information”. In: *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. CIKM ’15. Melbourne, Australia: Association for Computing Machinery, 2015, pp. 891–900. ISBN: 9781450337946. DOI: 10.1145/2806416.2806512.

[29] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. “DeepWalk: Online Learning of Social Representations”. In: *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’14. New York, New York, USA: Association for Computing Machinery, 2014, pp. 701–710. ISBN: 9781450329569. DOI: 10.1145/2623330.2623732.

[30] Palash Goyal and Emilio Ferrara. “Graph embedding techniques, applications, and performance: A survey”. In: *Knowledge-Based Systems* 151 (July 2018), pp. 78–94. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2018.03.022.

[31] Janu Verma et al. “Heterogeneous Edge Embeddings for Friend Recommendation”. In: *CoRR* abs/1902.03124 (2019). arXiv: 1902.03124. URL: http://arxiv.org/abs/1902.03124.

[32] Joan Bruna et al. “Spectral Networks and Locally Connected Networks on Graphs”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2014. URL: http://arxiv.org/abs/1312.6203.

[33] Federico Monti, Michael M. Bronstein, and Xavier Bresson. “Geometric Matrix Completion with Recurrent Multi-Graph Neural Networks”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3700–3710. ISBN: 9781510860964.

[34] William L. Hamilton, Rex Ying, and Jure Leskovec. “Inductive Representation Learning on Large Graphs”. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS’17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1025–1035. ISBN: 9781510860964.

[35] Yujia Li et al. “Gated Graph Sequence Neural Networks”. In: *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2016. URL: http://arxiv.org/abs/1511.05493.

[36] Claire Longo. *Evaluation Metrics for Recommender Systems*. https://towardsdatascience.com/evaluation-metrics-for-recommender-systems-df56c6611093. [Online; last accessed 17-Mar-2020]. 2018.

[37] Eugene Yan. *Serendipity: Accuracy's unpopular best friend in Recommender Systems*. https://towardsdatascience.com/serendipity-accuracys-unpopular-best-friend-in-recommender-systems-ca079b493f3c. [Online; last accessed 17-May-2020]. 2018.

[38] Sean M. McNee, John Riedl, and Joseph A. Konstan. "Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems". In: *CHI '06 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '06. Montréal, Québec, Canada: Association for Computing Machinery, 2006, pp. 1097–1101. ISBN: 1595932984. DOI: 10.1145/1125451.1125659. URL: https://doi.org/10.1145/1125451.1125659.

[39] Wikipedia. *'Kurtosis'*. https://en.wikipedia.org/wiki/Kurtosis#/media/File:Standard_symmetric_pdfs.svg. [Online; last accessed 08-Aug-2020]. 2020.

[40] Maruti Techlabs. *'5 Advantages Recommendation Engines can Offer to Businesses'*. https://towardsdatascience.com/5-advantages-recommendation-engines-can-offer-to-businesses-10b663977673. [Online; last accessed 20-Aug-2020]. 2017.

Appendix A

Supplementary information

Github link to software code repository: `https://github.com/HarshaHN/friendship_recommendation_for_gofrendly`