ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

ENRICHING PREDICTIVE MODELS USING GRAPH EMBEDDINGS

M.Sc. THESIS

Yaren YILMAZ

Department of Computer Engineering

Computer Engineering Programme

JANUARY 2023

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

ENRICHING PREDICTIVE MODELS USING GRAPH EMBEDDINGS

M.Sc. THESIS

Yaren YILMAZ
(504201541)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Prof. Dr. Şule GÜNDÜZ ÖĞÜDÜCÜ

JANUARY 2023

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

TAHMİNLEME MODELLERİNİN ÇİZGE GÖMMELERİ KULLANILARAK ZENGİNLEŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Yaren YILMAZ
(504201541)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Şule GÜNDÜZ ÖĞÜDÜCÜ

OCAK 2023

Yaren YILMAZ, a M.Sc. student of ITU Graduate School student ID 504201541 successfully defended the thesis entitled "ENRICHING PREDICTIVE MODELS USING GRAPH EMBEDDINGS", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Şule GÜNDÜZ ÖĞÜDÜCÜ** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Yusuf YASLAN** Istanbul Technical University	.....
	Assoc. Prof. Dr. Murat Can GANİZ Marmara University	.....
		.....

Date of Submission : **30 December 2022**
Date of Defense : **26 January 2023**

To my beloved family,

FOREWORD

Firstly, I would like to express my respect and gratitude to my advisor Prof. Dr. Şule Gündüz Öğüdücü for her support in my academic studies. I will be grateful for her assistance, motivation, and help forever.

I would like to express my deepest gratitude to my family and my boyfriend for their love, trust, understanding, and every kind of support not only throughout my thesis but also throughout my life. I would not be able to manage the hardships in my life without their existence.

I also want to thank my wonderful friends and colleagues, Kıymet Kaya and Öznur İlayda Yılmaz for their friendship and their help whenever I need it. I appreciate their lovely comments and encouragement throughout my master's degree.

Thanks to all my friends, relatives, and colleagues who have helped me during my master's education.

January 2023

Yaren YILMAZ
Computer Engineer

TABLE OF CONTENTS

ABBREVIATIONS

XGBR	**:** Extreme Gradient Boosting Regressor
LSTM	**:** Long Short-term Memory
BERT	**:** Bi-directional Encoder Representation
MAE	**:** Mean Absolute Error
HR	**:** Hit Ratio
NDCG	**:** Normalized Discounted Cumulative Gain
ARIMA	**:** Auto-regressive Integrated Moving Average
SARIMA	**:** Seasonal Auto-regressive Integrated Moving Average
SVR	**:** Support Vector Regression
GNN	**:** Graph Neural Network
GAN	**:** Graph Attention Network
RNN	**:** Recurrent Neural Network
NN	**:** Neural Network
HCA	**:** Hierarchical Cluster Analysis
GRU	**:** Gated Recurrent Unit
CF	**:** Collaborative Filtering
KG	**:** Knowledge Graph
BFS	**:** Breadth First Search
UMAP	**:** Uniform Manifold Approximation and Projection
TF-IDF	**:** Term Frequency-Inverse Document Frequency
c-TF-IDF	**:** Class Based Term Frequency Inverse Document Frequency
DBSCAN	**:** Density-Based Spatial Clustering of Applications with Noise
HDBSCAN	**:** Hierarchical Density-Based Spatial Clustering of Applications with Noise
MLP	**:** Multi-Layer Perceptron
MF	**:** Matrix Factorization
NeuMF	**:** Neural Matrix Factorization
DAE	**:** Denoising Auto-Encoder
NGCF	**:** Neural Graph Collaborative Filtering

SYMBOLS

$\mathbf{N}$	: Number of data points
$\mathbf{K}$	: Number of cluster
$\mathbf{X=\{x_1, x_2, ..., x_N\}}$	: Data points
$\mathbf{C=\{C_1, C_2, ..., C_K\}}$	: Set of clusters
$\mathbf{\mu_k}$	: Mean of all data points in cluster k
$\mathbf{r_{nk}}$	: indicator for the membership of the data point n to cluster k
$\mathbf{N_w}$	: Number of weeks
$\mathbf{W=\{w_1, w_2, ..., w_{N_w}\}}$	: Set of weeks
$\mathbf{N_p}$	: Number of products
$\mathbf{P=\{p_1, p_2, ..., p_{N_p}\}}$	: Set of products
$\mathbf{N_{iv}}$	: Number of invoices
$\mathbf{IV=\{iv_1, iv_2, ..., iv_{N_{iv}}\}}$	: Set of invoices
$\mathbf{M}$	: Number of recommendation
$\mathbf{N_i}$	: Number of items
$\mathbf{I=\{i_1, i_2, ..., i_{N_i}\}}$	: Set of items
$\mathbf{N_u}$	: Number of users
$\mathbf{U=\{u_1, u_2, ..., u_{N_u}\}}$	: Set of users
$\mathbf{N_t}$	: Number of topics
$\mathbf{T=\{t_1, t_2, ..., t_{N_t}\}}$	: Set of topics
$\mathbf{V=\{v_1, v_2, ..., v_{(N_i + N_u + N_t)}\}}$	: Vertices
$\mathbf{E}$	: Set of edges
$\mathbf{G}$	: (V, E)
$\mathbf{GX_i}$	: Node features for node i
$\mathbf{MP}$	: The meta-path scheme

LIST OF TABLES

LIST OF FIGURES

ENRICHING PREDICTIVE MODELS USING GRAPH EMBEDDINGS

SUMMARY

Today, artificial intelligence and machine learning models are developing rapidly and their usage areas are increasing in proportion to this speed. These developments have led to an increase in forecasting models in various industries. In the digitalizing, changing, and rapidly increasing data world, the spread of these solutions can be seen as a result of the moves made by companies to seize their competitive power and increase their profits. Especially with the increase in the amount of data, the need of the prediction models has also increased.

Predictive models are used in various fields. For example, in financial markets, forecasting models are used to predict future price movements. Also, in the healthcare industry, predictive models are used in the diagnosis and treatment of diseases. In a graph structure such as social networks, prediction models can be used to predict users' interests. Especially in the field of e-commerce, where social networks are seen a lot, data is often represented as a graph structure.

Graph embeddings can be used in a variety of prediction models. For example, when building a prediction model, we can improve the performance of the model by using graph embeddings. This helps the model better understand the relationships in the graph structure so it can make more accurate predictions. Also, graph embeddings can be used to predict relationships between nodes in a graph structure. For example, in a graph structure such as a social network, graph embeddings can be used to predict friend relationships between users. Demand forecasting systems and recommendation models are two different types of forecasting models. Demand forecasting systems are the accurate calculation of the future demand for a product. Graphs can be useful demand forecasting for several reasons. For example, demand prediction often involves working with sparse data, where there are many possible products or customers but only a small subset of them have any observed demand. Graphs can help to handle this sparsity by leveraging the relationships between entities to infer demand for products or customers that have not been directly observed. Graphs can be also used to handle the temporal dynamics of demand, such as how demand changes over time. Graphs can be very helpful to capture the compex interactions between different factors, such as customer preferences. Recommendation systems, on the other hand, are softwares used to make recommendations based on the interests of users. For example, in a music player app, by reviewing the songs a user has listened to and the artists they've listened to, other songs can be suggested that match that user's interests. Since recommendation systems generally work by finding the user or product similarities in data sets, using graph embeddings for performance improvement in recommendation systems can significantly improve performance. As a result, data can

be easily represented in graph structures in demand forecasting and recommendation systems. Graph embeddings from these graphs are used as an important method in demand forecasting systems and recommendation systems and can help improve the performance of forecasting models in these areas.

This thesis aims to use graph structures in demand forecasting and recommendation systems and to improve models with embeddings obtained from these graphs. This study generally consists of 2 parts: The use of graph embeddings in demand forecasting and the use of graph embeddings in recommendation systems.

In demand forecasting, which is the first stage, although the success of cutting-edge machine learning and deep learning models in demand forecasting has come to the fore according to recent research, the usage of datasets that are enriched using graph-based feature representations to improve demand forecasting models, is still rare. Therefore, in the demand forecasting stage, a model that predicts demand using graph embeddings is proposed. Unlike most existing methods, sales information data is used to extract various relationships between products, and these various relationships are used to create graphs. Five different embeddings are evaluated to reflect the different relationships between the products. These five different embeddings are obtained using five different graphs created using five different relationships. In order to obtain product embeddings, two different graph embedding methods named Node2Vec and GraphSAGE were preferred. After the product embeddings are obtained, Long short-term memory (LSTM) and Extreme gradient boosting (XGBoost) models are tried to perform demand forecasting. The Mean absolute error (MAE) metric was preferred to test the accuracy of the predicted values and the model was tested using a publicly available retail sales dataset. By running both models with different parameters, the best parameters were found and the results were obtained by using the best parameters while taking the final results of the models. According to the results, the use of graph embeddings increased the prediction success in both models. The prominent model came into prominence as the model formed by the Node2Vec graph embedding method and the LSTM method with the value of MAE=2.98.

Another study is the use of graph embeddings in recommendation systems. The model proposed in this study obtains graph embeddings for each product using the product title information. At this stage, a model used in natural language processing called BERT was used. With the proposed model, one embedding was created for each product using product titles, and category (topic) information was obtained for each product using the embeddings from the BERTopic model. This category information obtained is then used in the the next stage and enriches the user-product bipartite graph by transforming it into a heterogeneous user-product-topic tripartite graph. Unlike most existing methods, categories are created using only product titles as attributes, making it an easy-to-implement methodology that can be applied in real-world systems even if items don't have detailed item descriptions or comments. In addition, it is aimed to solve the sparsity problem in recommendation systems by learning the node embeddings using the meta-path-based Metapath2Vec algorithm to explore the resulting tripartite graph.

Two different categories of the Amazon dataset were used to observe the performance of the proposed model, and the results showed that the model outperformed even

the closest basic result by 3.8% when evaluated for both datasets according to hit raito. Further analysis confirms that incorporating category information into the graph structure provides an increase in recommendation accuracy, especially when recommending long-tail items. As a result, the proposed model has been evaluated using small versions of datasets and versions where we call cold-start products with little relevance, and the proposed model outperformed the base models for both hit rate and NDCG.

TAHMİNLEME MODELLERİNİN ÇİZGE GÖMMELERİ KULLANILARAK ZENGİNLEŞTİRİLMESİ

ÖZET

Günümüzde, yapay zeka ve makine öğrenmesi modelleri hızla gelişmekte ve kullanım alanları da bu hızla orantılı şekilde artmaktadır. Bu gelişmeler, çeşitli sektörlerde tahminleme modellerinin artmasına neden olmuştur. Dijitalleşen, değişen ve hızla artan veri dünyasında, bu çözümlerin yaygınlaşması, şirketlerin rekabet gücünü ele geçirmek ve karlarını arttırmak adına yaptıkları hamlelerin bir sonucu olarak görülebilir. Özellikle veri miktarının artmasıyla birlikte, tahminleme modellerinin doğruluk oranları da artmıştır.

Tahminleme modelleri, çeşitli alanlarda kullanılmaktadır. Örneğin, finansal piyasalarda, gelecekteki fiyat hareketlerini tahmin etmek için tahminleme modelleri kullanılır. Ayrıca, sağlık sektöründe, hastalıkların teşhisi ve tedavisinde tahminleme modelleri kullanılır. Sosyal ağlar gibi bir çizge yapısında ise, kullanıcıların ilgi alanlarını tahmin etmek için tahminleme modelleri kullanılabilir. Özellikle sosyal ağların çok fazla görüldüğü e-ticaret alanında veri sıklıkla çizge yapısı olarak temsil edilmektedir.

Çizge gömmeleri, çeşitli tahmin modellerinde kullanılabilir. Örneğin tahmin modeli oluştururken, çizge gömmeleri kullanarak modelin performansını iyileştirebiliriz. Bu, modelin çizge yapısındaki ilişkileri daha iyi anlayabilmesine yardımcı olur ve böylece daha doğru tahminler yapabilir. Ayrıca, çizge gömmeleri, bir çizge yapısındaki düğümler arasındaki ilişkileri tahmin etmek için de kullanılabilir. Örneğin, sosyal ağlar gibi bir çizge yapısında, bir kişinin arkadaşlarının arkadaşlarını tahmin etmek için çizge gömmeleri kullanabilir. Tahminleme modellerini, talep tahmin sistemleri ve öneri sistemleri olarak 2 ayrı sistem açısından ele alabiliriz. Talep tahmini sistemleri, bir ürünün gelecekte oluşabilecek talebinin, doğru olarak hesaplanmasıdır. Örneğin, bir e-ticaret sitesinde, bir kullanıcının hangi ürünleri almayı tercih ettiğini tahmin etmek için çizge gömmeleri kullanılabilir. Bu sayede, müşterilerin geçmiş alışverişleri ve diğer özelliklerine göre onların hangi ürünleri almayı tercih edeceklerini tahmin edebilir ve böylece stok yönetimi ve ürün çeşitliliği gibi konularla ilgili daha doğru karar verilebilir. Öneri sistemleri ise, kullanıcıların ilgi alanlarına göre önerilerde bulunmak için kullanılan yazılımlardır. Örneğin, bir müzik çalma uygulamasında, bir kullanıcının dinlediği şarkıları ve dinlediği sanatçıları inceleyerek, o kullanıcının ilgi alanlarına uygun diğer şarkılar önerilebilir. Öneri sistemleri genellikle veri setlerinde kullanıcı veya ürün benzerliklerini bularak çalıştığı için çizge gömmelerinin öneri sistemlerinde performans iyileştirmesi için kullanılması performansı ciddi oranda iyileştirebilmekledir. Sonuç olarak, talep tahmini ve öneri sistemlerinde veri kolayca çizge yapıları ise temsil edilebilmektedir. Bu çizgelerden elde edilen çizge gömmeleri, talep tahmin sistemleri ve öneri sistemlerinde önemli bir yöntem olarak

kullanılmaktadır ve bu alanlarda tahminleme modellerinin performansını iyileştirmeye yardımcı olabilir.

Bu tezde, yukarıda belirtildiği gibi talep tahmini ve öneri sistemlerinde çizge yapılarının kullanılması ve bu çizgelerden elde edilen gömmeler ile modellerin iyileştirilmesi hedeflenmiştir. Bu çalışma genel olarak 2 parçadan oluşmaktadır: Talep tahmininde çizge gömmelerinin kullanılması ve öneri sistemlerinde çizge gömmelerinin kullanılması.

İlk aşama olan talep tahmininde, son araştırmalara göre, talep tahmininde son teknoloji makine öğrenimi ve derin öğrenme modellerinin başarısı ön plana çıksa da, talep tahmin modellerini iyileştirmek için çizge tabanlı özellik temsillerini kullanarak veri kümelerinin zenginleştirildiği çalışmalar hala nadirdir. Bu yüzden talep tahmini aşamasında, çizge gömmeleri kullanılarak talebi tahmin eden bir model önerilmiştir. Mevcut yöntemlerin çoğundan farklı olarak, satış bilgisi verileri ürünler arası çeşitli ilişkileri çıkarmak için kullanılır ve çizgeler oluşturmak için bu çeşitli ilişkiler kullanılır. Ürünlerin farklı ilişkilerini yansıtmak için beş farklı gömme değerlendirilir. Bu beş farklı gömme, beş farklı ilişki kullanılarak oluşturulmuş beş farklı çizge kullanılarak elde edilir. Ürün gömmelerini elde etmek için Node2Vec ve GraphSAGE adında iki farklı çizge gömme elde etme yöntemi tercih edilmiştir. Ürün gömmeleri elde edildikten sonra, talep tahminini gerçekleştirmek için Uzun süreli kısa bellek (LSTM) ve Ekstrem gradyan arttırma (XGBoost) modelleri denenmiştir. Tahmin edilen değerlerin doğruluğunu test etmek için Ortalama mutlak hata (MAE) metriği tercih edilmiştir ve halka açık olan açık bir perakende satış veri seti kullanılarak model test edilmiştir. Her iki model de farklı parametreler ile çalıştırılarak en iyi parametreler bulunmuş ve modellerin son sonuçları alınırken en iyi parametreler kullanılarak sonuçlar alınmıştır.Sonuçlara göre, çizge gömmelerinin kullanımı her iki modelde de tahmin etme başarısını arttırmıştır. Öne çıkan model MAE=2.98 değeri ile Node2Vec çizge gömülüm yöntemi ve Uzun süreli kısa bellek yönteminin oluşturduğu model olarak ön plana çıkmıştır. Burada önerilen çözüm, öncelikle marka veya kategori gibi ürünlerin doğal bir topolojisi olmadığında bile sadece önceki satış verisini kullanarak daha doğru bir talep tahmini yapmayı hedefliyor. Bununla beraber, çeşitli çizgeler oluşturarak elde edilen ürün gömmeleri ile, sınırlı satış bilgisine sahip ürünlerin sebep olduğu veri seyrekliğinin etkisini de azaltmayı hedefliyor.

Bir diğer çalışma ise çizge gömmelerinin öneri sistemlerinde kullanılmasıdır. Bu çalışmada önerilen model, her ürün için ürün başlık bilgilerini kullanarak birer çizge gömmeleri elde eder. Bu aşamada BERT adında doğal dil işleminde kullanılan bir model kullanıldı. Önerilen model ile, ürün başlıkları kullanarak her ürün için birer gömme edildi ve bu gömmeler kullanılarak da BERTopic modeli kullanılarak her ürün için bir kategori bilgisi elde edilmiştir. BERTopic yöntemi ile kategori bilgilerinin elde edilmesi konu modelleme olarak kabul edilebilir. Elde edilen bu kategori bilgileri daha sonra çizge aşamasında kullanılır ve heterojen kullanıcı-ürün çizgesini kullanıcı-ürün-konu üçlü çizgesine dönüştürerek zenginleştirir. Mevcut yöntemlerin çoğundan farklı olarak kategoriler, özellik olarak yalnızca ürün başlıkları kullanılarak oluşturulur, bu da onu, maddelerin açıklama metinleri veya yorumları olmasa bile gerçek dünyadaki sistemlerde uygulanabilen, uygulaması kolay bir metodoloji haline getirir. Ayrıca, elde edilen üçlü çizgeyi keşfetmek için meta-yol tabanlı Metapath2Vec

algoritması kullanarak düğüm gömmelerini öğrenerek öneri sistemlerindeki seyreklik probleminin çözülmesi hedeflenmiştir. Önerilen modelin performansını gözlemlemek için Amazon veri setinin iki farklı kategorisi kullanıldı ve sonuçlar, isabet oranına göre her iki veri seti için değerlendirildiğinde modelin en yakın temel sonuçtan bile %3.8 daha iyi performans sergilediğini göstermiştir. Yapılan diğer analizler, kategori bilgilerinin çizge yapısına dahil edilmesinin, özellikle uzun kuyruklu öğeler tavsiye edilirken tavsiye doğruluğunda bir artış sağladığını doğrular. Sonuç olarak, önerilen model, veri kümelerinin küçük versiyonları ve soğuk başlatma adını verdiğimiz az ilişkisi olan ürünlerin alınarak oluşturulduğu versiyonları kullanarak değerlendirilmiştir ve önerilen model hem isabet oranı hem de NDCG için temel modellerden daha iyi performans göstermiştir. Bu öneri sisteminde geliştirilen çözüm, öncelikle öneri sistemlerinde konu modelleme yönteminin faydasını vurgulamaktadır. Model, herhangi bir müşteri ile ilişkisi olmayan ürünler için bile farklı meta-yollar tanımlayarak veride seyreklik sorununu hafifletmeyi amaçlamaktadır. Ayrıca konu modellemesi yöntemi, aykırı olarak belirlenen konusuz ürünleri de kendi gömmesine en yakın gömmeye sahip konuya atarak geliştirilmiştir.

İlerleyen çalışmalarda, öncelikle iki model de daha fazla veriseti kullanılarak test edilebilir. Daha fazla veriseti kullanımı, her iki modelde de, sadece perakende sektöründe değil, işlemsel verilerin olduğu farklı alanlarda da modellerin kullanılabilirliğinin anlaşılmasına yardımcı olacaktır. Ayrıca talep tahmininde, model şimdilik tek bir değerlendirme metriği kullanılarak değerlendirilmektedir. Daha fazla değerlendirme ölçütü, yaklaşımın etkililiğini farklı bakış açılarından anlamaya yardımcı olur. Öneri sistemi için model, uygulamaya daha fazla açıklanabilirlik katmak için vaka çalışmaları genişletilebilir. Ayrıca, ürünlerin başlıkları kullanılarak ürünler için elde edilen kategoriler gibi kullanıcılar için de benzer şekilde kullanıcı yorumları kullanılarak kategoriler elde edilebilir. Kullanıcıların ürünlere yaptığı yorumların bilgisi modele eklenirse daha anlamlı gömmeler elde edilebilir. Son olarak, tezin tamamı için, farklı çizge gömme sinir ağı modellerinin kullanılması daha iyi sonuçlar gözlemlememize yardımcı olabilir.

1. INTRODUCTION

Demand prediction and recommendation systems are two important tools that businesses use to better understand their customers and make informed decisions about production and marketing strategies. Demand prediction involves forecasting the future demand for a product or service based on past data and trends. This can help businesses optimize their production and inventory management to meet customer demand. A recommendation system, on the other hand, is a tool that suggests items to users based on their previous interactions or preferences. This can help businesses increase customer satisfaction and loyalty by offering personalized recommendations that are tailored to each individual's interests and needs. Both demand prediction and recommendation systems have become increasingly important in the digital age, as the amount of data available to businesses has grown exponentially and the need to analyze it has become more pressing.

In recent years, graph embeddings have become an increasingly popular technique for representing nodes in a graph as continuous vectors, which can then be used as input for predictive models [2]. Graph embeddings can be a valuable source of context and details about the underlying patterns and properties of the graph by capturing the structure and connections between the nodes in the graph such as the connections between friends in social networks or the co-occurrence of words in a document. This makes them particularly useful for predictive models such as demand prediction and recommendation systems that rely on understanding the relationships between different entities.

The use of graph embeddings in predictive models has grown significantly in recent years due to their ability to capture complex relationships and patterns in data [3]. For example, in a demand prediction system, graph embeddings can be used to represent the relationships between different products, customers, and purchasing patterns [4], enabling the predictive model to make more accurate predictions about future demand.

Similarly, in a recommendation system, graph embeddings can be used to represent the relationships between different users and items [5], allowing the predictive model to recommend relevant and personalized items to users.

One of the key benefits of using graph embeddings in predictive models is their ability to handle large and complex graphs with millions of nodes and edges. By representing nodes as n-dimensional continuous vectors, graph embeddings can reduce the dimensionality of the data and make it more manageable for predictive models to handle. In addition, graph embeddings can capture the structure and relationships between nodes in a way that is not possible with traditional techniques, enabling predictive models to make more accurate and reliable predictions [5].

1.1 Purpose of Thesis

This thesis aims to explore the use of graphs and graph embeddings as a means of improving the performance of predictive models. In this thesis, a study was conducted on developing demand forecasting and recommendation systems forecasting models in particular.

The focus of this thesis is to explore different techniques for constructing graphs and computing graph embeddings and to evaluate their impact on the performance of predictive models in two different application domains which are demand prediction and recommendation systems.

Focusing on this purpose, the following research questions were investigated within this study:

1. How do different graph embedding techniques perform for computing graph embeddings in demand prediction and recommendation systems in terms of accuracy and efficiency?

2. How can graph embeddings be used to improve the performance of predictive models in different application domains, recommendation systems, and demand prediction?

3. How do different parameters and configurations of graph embedding algorithms impact the quality of the resulting embeddings and the performance of predictive models?

4. How do different types of relationships between nodes impact the quality of the embeddings?

To answer these questions, a detailed analysis of various techniques is conducted for computing graph embeddings and their performance is evaluated on a range of recommendation and demand prediction tasks. The use of graph embeddings is also explored in different application domains and the impact of different types of relationships between nodes is analyzed on the quality of embeddings.

1.2 Contribution of Thesis

In this work, the effectiveness of graph embeddings for predictive models is examined. Detailed information on the proposed models is given in Section 3 and Section 4.

The contributions of the studies in this thesis are two folds:

- In Section 3, the proposed solution demonstrates how to use different relationships between items to enrich previous sales data for more accurate demand prediction even when there is no natural graph topology like brand or category. By employing the representation of items by creating various graphs, it is possible to reduce the effect of data sparsity with limited previous sale information.

- In Section 4, the proposed solution emphasizes the benefit of topic modeling to improve the graph embeddings for recommendation systems. The model aims to alleviate the sparsity problem by defining different metapaths to learn that have no relation to any customer. The topic modeling is also improved to find the most appropriate topics for each item.

1.3 Structure of Thesis

In Section 2, related works are given. The thesis includes two different studies to show the usage of graph embeddings in predictive models. That's why, both Section 3 and Section 4 shows the proposed models, the dataset, and experimental results for demand prediction and recommendation systems applications, respectively. Section 5 summarizes the work and makes recommendations about future works.

2. RELATED WORKS

This chapter summarizes previous works related to the thesis. Related studies are given under three main headings: Demand Forecasting, Recommendation System, and Graph Embedding.

2.1 Demand Forecasting

Demand forecasting is a crucial aspect of supply chain management, as it allows businesses to optimize their production and distribution processes to make informed decisions about inventory management and pricing [6].

In-depth research has been done in this area, which has a vast literature. At the beginning of the implementation of demand forecasting, traditional time-series methods such as exponential smoothing [7,8], Auto-regressive Integrated Moving Average (ARIMA) [9,10], and Seasonal Auto-regressive Integrated Moving Average (SARIMA) [11] were quite popular. With the development of machine learning and deep learning models which are based on artificial intelligence, the success of these models compared to traditional models has been proven by numerous studies [12,13]. The study [13] shows a comparative study of SARIMA and Long-Short Term Memory (LSTM) to predict monthly product demand, and LSTM outperforms in predicting short-term prediction.

A study [14] proposes an ensemble model with nine different time series models which are Support Vector Regression (SVR), and deep learning models using boosting ensemble strategy. Another study [15] aims to make better estimations for products with no sales using a newly proposed model combined with two different methods. The first method called CombTSB decides which model should be selected to implement the set of pre-processed time-series data. Then, the second method called Clust-Avg uses a clustering algorithm, K-Means Clustering, to estimate demand for new items. A deep neural framework that formulates the issue as a sequence-to-sequence is

proposed in the article [16]. Qi et al. build a Gated Recurrent Unit-based (GRU-based) encoder-decoder architecture to include the promotion campaign for the target product.

As far as knowledge goes, the use of graph embeddings or graph neural networks (GNNs) in demand forecasting is not very common. Moreover, to the best of out knowledge, there is no study that analyzes the effect of different graph embeddings for demand prediction. One of the studies based on graph representation [17] uses graph representation to segment demand prediction in e-commerce where the aim is to improve the demand prediction accuracy of market segments with limited records by transferring the knowledge from mainstream segments. The authors create a unique algorithm with two GRU for collecting both local and seasonal temporal trends in order to uncover the complicated linkages and increase the stability. Shi et al. additionally create two distinct data-driven and knowledge-segmented graphs. User transaction behavior is employed with the DeepWalk technique as the segment knowledge.

Another work [18] aims to forecast the demand of products using their natural topology such as category and brand using GNN. According to Liao et al., the demand for one product within the same category or brand may have an impact on the demand for a different product, or give us an idea about the demand for a different product. Their proposed model uses Graph Attention Network (GAN) to learn the structural information between products and GRU to capture the temporal patterns in the time-series dataset.

In contrast to the studies stated above, a graph is constructed using the relationship matrix of products only from sales information. In order to train a machine learning model to predict product demand, feature embedding learning is added to the sales data. Additionally, obtaining a compressed representation in the form of embeddings enables us to accurately estimate demand even with low-order models. Finally, even for limited sales information, employing feature representations undoubtedly enhances the demand forecast model.

2.2 Recommendation Systems

There are several methods of developing recommendation systems, including Collaborative Filtering (CF) [19]–[22], content-based filtering [23,24], and hybrid methods [25]–[27]. CF approaches build a model from a user's past behavior as well as similar decisions made by other users. This model is then used to predict items that the user may have an interest in. Content-based filtering methods utilize a series of discrete characteristics of an item in order to recommend additional items with similar properties. Hybrid methods combine collaborative filtering and content-based filtering approaches.

Most of the first research studies on recommendation systems were based on the CF method [28], which uses similarities between users and items. One of the limitations of CF-based models is that these models are not able to incorporate side features of items/users and generalize well in the task of cold start cases where there are no or very few interactions between users and items.

In recent years, GNNs have been more popular in recommender systems due to their capability to utilize the relational information among users and items. Various studies exist in the literature that use GNNs to improve recommender systems. GNNs can be implemented on the graphs that reflect the collaborative information between items and users as user-item graphs [29]–[31], or are constructed as Knowledge Graphs (KGs) that show a topology of the given data. Since the side information is essential in recommender systems to handle the cold strat problem, KGs have been used as helpful side information to improve the recommendation performance [32]–[34].

The study proposed in [35] aims to learn user and item representations by using high-order connectivity in the user-item graph. They demonstrate the importance of explicitly exploiting the collaborative signal in recommender systems, and use an embedding layer to initialize user and item embeddings, multiple embedding propagation layers to obtain high-order connectivity relations, and a prediction layer. The model learns the embedding by aggregating the embedding and optimizing the affinity score of a user-item pair. Another study [36] aims to learn the hierarchical

representations on a user-item bipartite graph. They use multiple GNN modules and a clustering algorithm. By using a hierarchical fashion, the model can learn user preferences and item attractiveness in a hierarchical way.

In more recent studies, hybrid recommender systems are gaining in popularity by combining both collaborative and content-based approaches to solve cold start problems. Because, in addition to user-item representations, the usage of item or user information can help to improve recommender systems and help to enrich the information obtained from the user-item representations. Due to these reasons, they have become undoubtedly important sources for recommender systems.

In a recent study [37], a hybrid recommender system framework is proposed using both graph embeddings and contextual word representations. By using two different representations, they aim to learn both collaborative and content-based features.

Another concept in recommendation systems is meta-paths which help recommendations systems to tackle the information in a network [38]–[40]. As for the attempts to use meta-paths in recommendation systems, Anwaar et al. [41] propose a meta-path and entity-aware graph neural network for the recommendation. They collect information from different meta-paths and combine the information from multiple meta-paths using the attention mechanism. Despite they take benefits of the meta-paths, the usage of entities is not applicable in the case when there is no entity information for the users or items.

Different from previous models, in this thesis a novel recommendation model is proposed for providing item recommendations, where we first obtain item sub-categories from item titles. Using only title information of items makes it possible to apply the approach on any platform easily even if text data for items or users such as review or comment texts are not available (or not reliable). We then build a heterogeneous tripartite graph to learn joint user and item embeddings that capture more accurately user interests in items. By successfully addressing the cold start challenges in our novel model, our proposed model results in performance improvements over existing methods, as demonstrated through experimental tests.

3. ENRICHING DEMAND PREDICTION WITH PRODUCT RELATIONSHIP INFORMATION USING GRAPH EMBEDDINGS

This chapter explains a novel demand prediction model using the sales-based relationship between products. The sales-based relationships are constructed using graph structures. Then, the graphs are used to create graph embeddings which in turn are utilized for demand prediction.

3.1 Background Methods

This study uses various background methods for graph construction, graph embeddings, and prediction. The following subchapters summarise these methods.

3.1.1 Apriori algorithm

Apriori Algorithm [42] is used to find the association rules within the transaction data where association rules mean the relationships among the data items. The algorithm iteratively runs and determines the candidate items for the next step. The parameter *minimum_support* (3.1) is used to determine the candidate items for the next step where the support value of the $n-itemset$ is bigger than *minimum_support*. When $2-itemset$ occurs 2 times out of 10 transactions, the support value should be 0.2. In this study, the apriori algorithm helps us to find the product pairs and their frequency value for the graph construction step.

$$Support(A) = \frac{Number\,of\,transaction\,in\,which\,A\,appears}{Total\,number\,of\,transactions} \tag{3.1}$$

3.1.2 K-Means clustering

The unsupervised K-Means algorithm [43] aims to cluster N data points X into a fixed K number of disjoint clusters C where $C = \{C_1, C_2, ..., C_K\}$. The objective function (Equation 3.2) is the *sum of squares distances* of each data point to its cluster center and the goal is to minimize the value J. In Equation 3.2, $r_{nk} \in 0, 1$ is an indicator

variable for each data point, that indicates the membership of that data point to cluster k, and μ_k is the mean of all points in cluster k. These values are the output of the clustering process.

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|x_n - \mu_k\|^2 \tag{3.2}$$

The algorithm tries to minimize the *within_cluster_sum_of_squares* (Equation 3.2) where N is the number of data points and K is the number of clusters. K-Means clustering aims to find μ_k and t_{nk} iteratively. Each iteration consists of two main steps:

1. The first step is the expectation step (Equation 3.3). The goal is finding the r_{nk} that minimizes J for each x_n in X. It is aimed to assign the data point x_n to the closest cluster according to its sum of squared distance from the cluster's centroid μ_j.

$$r_{nk} = \begin{cases} 1 & if k = argmin_j \left\|x_n - \mu_j\right\| \\ 0 & otherwise. \end{cases} \tag{3.3}$$

2. The second step is the maximization step. In Equation 3.4, the numerator is the *Sum of all points in a cluster* and the denominator is the *Number of points in a cluster*.

$$\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}} \tag{3.4}$$

The iterations are stopped when J no longer changes.

3.1.3 Node2Vec algorithm

Node2Vec [44] is an algorithm for learning continuous-valued representations, in other saying embeddings of nodes in a graph. It is based on the idea of random walks, which consists of a sequence of steps taken by a random walker on a graph. Node2Vec implements random walks for a given node v of length l. Equation 3.5 shows the calculation of the probability of moving to node x from the node v where c_i is the ith node in the walk and π_{vx} is the unnormalized transition probability between nodes v and x, and Z is the normalizing constant. The formula finds a normalized probability of going to x from v, if there is an edge between the nodes v and x.

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{vx}}{Z} & \text{if}(v,x) \in E \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

The second step is understanding the biased random walk. The simplest way to use a bias for the random walks is the edge weights in the graph but only if the graph is a weighted graph. If there is an unweighted graph, the bias is created using two parameters p and q as shown in Equation 3.6 where d_{tx} is the. $2nd$ order random walk is used and it gives the flexibility of searching strategy by using the parameters the return parameter p and the in-out parameter q where p indicates the probability of a random walker getting back to the previous node and q indicates the probability that a random walker can pass through a previously unseen part of the graph. These two parameters p and q make the algorithm smoothly interpolate between Breadth-First Search (BFS) and Depth-First Search (DFS). Higher p values make the random walk closer to BFS and higher q values make the random walk closer to DFS. Figure 3.1 shows an example of a random walk procedure when the walk transitioned from t to v and walks to the next node from v.

$$\alpha_{pq}(t,x) = \begin{cases} \frac{1}{p} & \text{if} d_{tx} = 0 \\ 1 & \text{if} d_{tx} = 1 \\ \frac{1}{q} & \text{if} d_{tx} = 2 \end{cases} \tag{3.6}$$

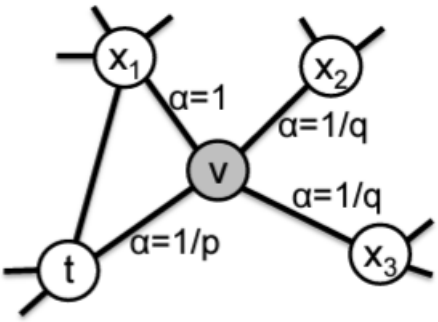


Figure 3.1 : Example of the biased random walk for node2vec

After the biased random walks are completed, the Word2Vec algorithm [45] is used to create node embeddings. The Word2Vec algorithm uses the Skip-Gram model. A skip-gram model is a neural network with a single input layer, a hidden layer, and an output layer and it aims to predict the probability of a given work being present when an input word is present. The hidden layer is responsible for transforming the input

word into a low-dimensional representation. In Node2Vec, we can assume the words as nodes. The biased random walks create a list of node ids like sentences. Then, these sentences (list of nodes) are used as inputs of the Skip-Gram model. In the training process, the model is presented with a target word and a set of context words, and it is trained to predict the likelihood of each context word given the target word. After the training process, the weights of the hidden layer can be used as the node embeddings.

3.1.4 Extreme gradient boosting regressor (XGBR)

XGBR [46] is an ensemble machine-learning algorithm that can be used for regression modeling. It is a decision tree-based ensemble model because decision trees are sequentially combined and weights are assigned to the outputs of individual trees. One of the main advantages of XBGR is that it assigns a higher weight to the misclassifications of the first decision tree and provides input to the next decision tree. In this way, the boosting method combines the multiple weak rules into one strong prediction rule and it aims to improve a single weak model by combining it with other weak models.

Since it is an iterative process, it is aimed to reduce the value of the objective function. The objective function of XGBR (Equation 3.7) includes both the training loss $L(\theta)$ and regularization term $\Omega(\theta)$. It is crucial to use the regularization term because it controls the complexity of the model and avoids overfitting by adding a penalty to the objective function. The common loss function used in XGBR is the squared error (Equation 3.8)

$$obj(\theta) = L(\theta) + \Omega(\theta) \tag{3.7}$$

$$L(\theta) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.8}$$

3.2 Proposed Model

Using the weekly demand data, the proposed model forecasts the sale amount of products $x_{\tau+1}$ at the day $\tau+1$ using the sale amounts of the previous days $X = x_1, x_2 \ldots, x_\tau$.

Figure 3.2 shows the general framework of the proposed model. Firstly, the online retail dataset is cleaned and preprocessed in the data preparation step. The dataset is used in two different steps which are demand prediction and graph construction. First, the transactional data are used for the graph construction step. The Apriori algorithm and K-Means algorithm are used to find the product pairs in the graph construction step and these product pairs are used as the edge set for the graphs. Then, these graphs are used as inputs for the graph embedding models. Two different graph embedding models are tried which are the Node2Vec algorithm and GraphSAGE. The next step is the demand prediction part. Both transactional data and product embeddings are used as input for the demand prediction model. The transactional data is converted into a time-series format using the window sliding method. XGBR and LSTM are used as the predictive models for the demand prediction part and the results are compared. According to the results, XGBR gives better results than LSTM. On the other hand, the Node2Vec algorithm produces better embeddings than the GraphSAGE model according to the MAE of the demand prediction model.

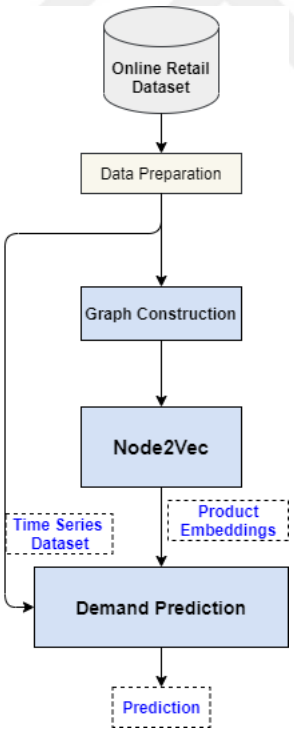


Figure 3.2 : General flow of the proposed demand prediction model [1]

3.2.1 Graph construction

The proposed demand forecasting model aims to do research on the effect of different graph embeddings on demand forecasting.

In this step, Apriori Algorithm and K-Means Clustering are applied to extract relationships among users/items for graph construction. Five different graphs are constructed by considering the different relationships between users and items.

$$Support_m(p_i,\ p_j) = \frac{freq_m(p_i,\ p_j)}{N_m} \tag{3.9}$$

Graph 1 is created based on the frequency of sales of products in the same week. Each product is a node in the graph. We put edges between products if they are sold in the same week. In Figure 3.3 vertically, products B and E are sold in the same week, it is expected that these two products have an edge between them.

The frequent 2-itemsets and their support values are found by using the Apriori algorithm. The support value is also used as the edge weight in the graph. A matrix $M_1 \in \mathbb{R}^{N_w \times N_p}$ is used as input for the Apriori algorithm. Then, the algorithm finds the support value for all the possible frequent 2-itemsets. In equation (3.9), m is the week, N_m is the number of weeks and the frequency $freq_m(p_i, p_j)$ indicates the number of weeks that p_i and p_j are sold in week m.

The graph is constructed as a weighted graph to research the effect of the frequency of items being bought together on the success of the demand prediction.

Graph 2 is constructed as a weighted graph based on the frequency of sales of products in the same invoice. When we examine Figure 3.3 horizontally, products A, E, and F are sold in the same invoice i_3, it means there should be edges between all of the possible frequent 2-itemsets in A, E, and F (A and E, A and F, E and F). The second graph is constructed using the same algorithm as the first graph. First, a matrix $M_2 \in \mathbb{R}^{N_I \times N_p}$ is created to use as the input for the Apriori algorithm using the transactional data. In the matrix M_2, each cell defines whether the product is sold in that invoice

or not. For the sake of example, if $M_2[i_j, p_i]$ is 1, it means p_i is sold in the invoice i_j. Then, the Apriori algorithm finds all the possible 2-itemsets and their support value.

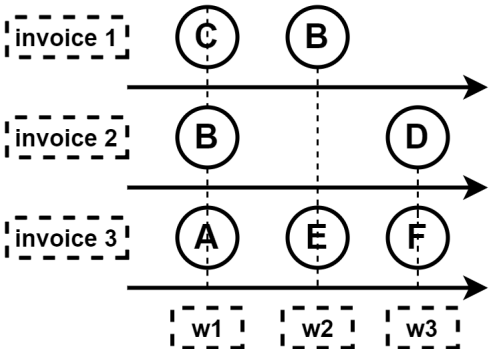


Figure 3.3 : Products sold in the same week and in the same invoice [1]

Graph 3 The next graph is constructed as a weighted graph based on the frequency of sales of products bought by the same customer. By creating this graph, the goal is to learn the customers' habits because we believe that if two products p_i and p_j are bought by the same customer, the demand for p_i might be similar to the demand for p_j. $M_3 \in \mathbb{R}^{N_c \times N_p}$ is constructed, where each cell defines if the product p_i is bought by the customer c_j. The Apriori algorithm again finds the frequent 2-itemsets and their support value to construct the graph.

Graph 4 is constructed using a similar idea in Graph 3 but it has a different structure. In Graph 3 there are edges between products if they are bought by the same customer. In Graph 4, there are edges between all the possible product combinations but only the corresponding edge weight changes. The euclidean similarity between all the products is obtained using the equation (3.10), and the weight between the products is calculated using matrix M3.

$$d\left(p_i, p_j\right) = \sqrt{\sum_{k=1}^{N_c} \left(M_3[k,i] - M_3[k,j]\right)^2} \tag{3.10}$$

Graph 5 is constructed using the cluster similarity between products where the product clusters are obtained via K-Means Clustering. The clusters are calculated using the best number of clusters K. Section 3.4.2 explains how the number of clusters K is determined. While creating graphs, some sale-based information is used which are price, sold by how many customers, and sold in how many invoices. The approach puts

edges between the products that are in the same cluster. Equation (3.11) shows how to assign weights to the graph. In order to connect the generated graph, unconnected clusters are then connected by determining the nearest cluster members from other clusters.

$$\omega_{i,j} = \begin{cases} 1 & \text{if } k_i = k_j \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

where k_i is the cluster of the product i and k_j is the cluster of the product j.

3.2.2 Demand prediction using XGBR

After all the graphs are implemented, the Node2Vec is used to construct product embeddings. Since its biased random walk feature, the Node2Vec algorithm is one of the best choices because it enables us to generate embeddings by learning more structural information. BFS-like exploration is used, which is effective at learning the network's topology, by increasing the q hyperparameter above 1. Finally, XGBR predicts the demand. The XGBR receives the graph embeddings as input along with other data like date and pricing features. One of the main reasons why XGBR is preferred over other methods is largely due to their sparsity awareness [46]. The dataset has a lot of 0 values because daily demand prediction is implemented and most products are not sold every day. These values are handled thanks to XGBR's awareness of its sparsity.

3.3 Dataset

The online retail dataset [1] includes $541,909$ transactions of 4070 products between 12/01/2010 and 12/09/2011. It contains transaction data from a UK-based online retail store for 373 days. This dataset is preferred because of its feature that allows us to construct relationships between products based on invoice and customer information. The preparation steps are split into two sections because some steps are implemented to make the dataset ready for the graph construction step and some steps are implemented to make the dataset ready for the time series prediction.

[1] https://archive.ics.uci.edu/ml/datasets/online+retail, December 2022

3.3.1 Data preparation for graph construction

The basic data preparation steps are applied. The first step is handling the missing values. There are 2 different columns with some missing values which are *CustomerID* and *Description*. The column *Description* is empty for 0.26% of the transactions and 24.92% of the column *CustomerID* is missing. As the missing value handling technique, it is preferred to drop the rows with missing values. Especially for the column *CustomerID*, it is important to have a value for each row, because some graphs are constructed using the relationships between products based on the customers' purchase habits and there is no appropriate way to fill the missing values of an ID column. After this step $135,686$ rows are deleted.

Another cleaning operation is done based on the explanation of the dataset. Since some transactions are canceled, these transactions are marked by adding a 'C' letter in the *InvoiceID* column in the dataset. The canceled transactions are 2.2% of the total transactions and these rows are dropped. In the beginning, the total number of products (the count of unique *StockCode*) was 4070. After the preprocessing steps, the total number of products decreased to 3663. After that, the products which have irrelevant *StockCode* values are removed and yield a decrease of 881 in the total number of products. The irrelevant *StockCode* means non-numeric codes that do not conform to the general stock format according to the dataset explanation.

The next preparation step is handling outlier values. After checking the statistical information of numerical columns, it is realized that the column *UnitPrice*, which represents the unit price of a product, has outlier values. While the third quartile of the column is 3.75, the maximum value is $649,5$. The handle outliers in this column, 99% of the column, which is equal to taking the log of the feature, are selected and the rest of them are assumed as outliers and they are deleted. The same outlier handling technique is applied for another column *Quantity*. Finally, the total number of products becomes 2748.

3.3.2 Data preparation for sale information

Before using the dataset for the time series prediction, the dataset is converted into time-series format using the window sliding approach. Using this approach, one can create a time-series dataset where the value of the last τ days can be used to predict the value of the $\tau + 1$th day. Figure 3.4 shows how the window sliding approach works. First, if τ is equal to 6, then the value of d_7 is predicted using the values of the last 6 days from d_1 to d_6. In the next step, the window slides by 1 day and the value of d_8 is predicted using again the previous 6 days' values from d_2 to d_7.

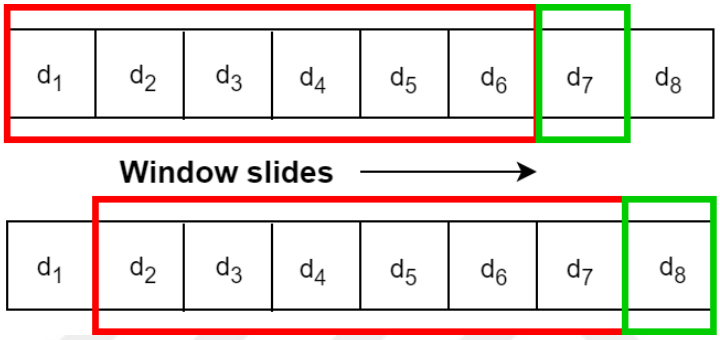


Figure 3.4 : The window sliding approach

In this project, the last 30 days of the sale information are used to predict the next day. That's why the first 30 days can't be predicted by the model. While converting the dataset into this format, 36 products are lost because these products are only sold in the first 30 days.

Another important preparation step is selecting products that are common to each graph. It is important that the time-series data only includes the products that are common to each graph. If there is a product in the time-series data and the same product is not a node in the graph, it is not possible to predict the demand for that product. As a result, only the transactions of products that are common to each graph are selected and the rest of the transactions are removed. This operation yields a decrease of 420 products the total number of products becomes 2328. These eliminated 420 products are also the products with less sales information than the others.

Besides the time-series data of the last t days, the following features are added to the dataset for the model input: *Quarter, Month, Week, Weekday, Day, DayofYear*, *sum, min, mean, max, median* values of the column *UnitPrice* by both weekly and daily.

3.4 Experiments

In the experiments, the results obtained using different graph embeddings are compared. The success of the embeddings is calculated using the demand prediction model.

The dataset is split into training and test sets. After ordering the transactional data by week, the first 70% of weeks are selected as the training set and the rest of the weeks are selected as the test set. The important point here is that the graphs are constructed only using the transactional data from the training set, the test set is never used during model development.

3.4.1 Evaluation metric

Mean Absolute Error (3.12) is used as the evaluation metric. It is a metric used to evaluate the regression models and shows how accurate the predictions are. Firstly, the absolute error which is the difference between the predicted value $\hat{y}_i$ and true value y_i is calculated for each data point i and then all the absolute errors are summed. The MAE of the model is calculated by dividing the sum by the number of total data points n.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.12}$$

3.4.2 Graph experiments

StellarGraph Library [47] is used for the graph experiments. As it is mentioned, the Apriori algorithm is used to create Graph 1, Graph 2, and Graph 3. In order to find the most appropriate graph structure for demand prediction, different minimum support values are used and the statistical information of different graphs is compared. Table 3.1, 3.2, and 3.3 show the statistical information of different graphs created with

different minimum support values. The statistical information includes the number of nodes V, is the graph connected or not is_C, and the connectivity value C. It is preferred to have a connected graph than an unconnected one because it is impossible to learn the detailed relationship information for a node when it is in the unconnected small subgraph. Assume that the graph is an unconnected graph and consists of 2 subgraphs where one subgraph includes the 99% of the relationships and the second subgraph only contains 2 nodes A and B and they are connected to each other. It is not expected that learn the complex relationships for node A or node B because they have only 1 edge. By choosing a connected graph instead of an unconnected graph, it is also aimed to reduce the number of nodes in the graph for faster calculations. That's why, a minimum support value of 0.0001 is chosen for Graph 2 (Table 3.2), and 0.001 is chosen for Graph 3 (Table 3.3). For the sake of the explanation, minimum support value 0.3 in Table 3.1 means that there will be an edge between products sold in at least 0.3 of the total number of weeks (at least in $0.3 \times 51 = 15$). The results show that the graphs created with bigger minimum support values tend to have fewer edges and be unconnected. It also results in a decrease in the number of products (nodes) in the graph. More minimum support values are tried and compared to their graph results, but it is tried to decrease the minimum support value as much as possible to create embeddings of as many products as possible. Finally, the chosen minimum support values for each graph are given as follows: 0.05 for Graph 1, 0.0001 for Graph 2, and 0.001 for Graph 3.

Table 3.1 : Graph 1

min_sup	V	is_C	C
0.3	1828	Y	0.77
0.2	2140	Y	0.79
0.1	2430	Y	0.83
0.05	2583	Y	0.88

Graph 4 includes all the products in the dataset because it is created based on the euclidean similarity between products.

The last graph construction experiment is applied to Graph 5. Since it is created using the K-Means Clustering, *Elbow Method* is used to find the best number of clusters.

Table 3.2 : Graph 2

min_sup	V	is_C	C
0.0007	1703	N	0.05
0.0005	1875	N	0.07
0.0002	2219	N	0.13
0.0001	2362	Y	0.19

Table 3.3 : Graph 3

min_sup	V	is_C	C
0.01	795	N	0.04
0.008	981	N	0.05
0.005	1314	N	0.09
0.001	2196	Y	0.27

Elbow Method calculates the Within-Cluster Sum of Square (WCSS) for different K values and plots the results where the x-axis shows different K values and the y-axis shows the WCSS corresponding to the K value. After that, the K value where you can see the elbow shape in the plot is chosen. For this dataset, the best K value is decided as 18. Some transactional information is used for the K-Means Clustering as follows: *MedianPrice, Mean Price, Difference between Maximum and Minimum Price, How many users bought the product, How many invoices the product was sold in total*. While building edges between products, the products in the same cluster are linked to each other. But after this process, Graph 5 consists of 18 different sub-graphs, because only the products those are in each cluster are connected. To make the graph connected, one node from each cluster is chosen that is closest to the center of the cluster, and these nodes from each cluster are linked to another node from the closest different cluster. The reason why the closest nodes to the center of the cluster are chosen from each cluster is that the nodes which are closest to the center of the cluster can reflect the general information of the cluster better than the nodes which are farthest nodes to the cluster center. Thanks to this operation, Graph 5 becomes a connected graph. It is believed that similar products according to their previous sale information might have similar demand.

The final step in the graph experiments is finding the intersected products in 5 different graphs. 2328 products are common in all of the graphs. That's why only transactions

of those 2328 products are used for the prediction task. In this way, we were able to compare the results obtained from different graphs and different graph embedding models.

3.4.3 Comparison models

The below models are used to compare the proposed model with them. LSTM is preferred because it captures more complex relationships in long-term time series data. LSTM doesn't use any graph structure, it is only a time series-based prediction model. It is used as the comparison model in the demand prediction part as shown in Figure 3.2.LSTM and XGBR are used as the prediction models. On the other hand, Node2Vec and GraphSAGE models are used for the graph embedding part. GraphSAGE is used as the comparison model for the graph embedding step. It is explained because the results obtained using Node2Vec embeddings are compared with the results obtained using GraphSAGE embeddings. GraphSAGE is chosen as a comparison model for graph embedding because it is able to use the information from the node features and can learn better embeddings than the Node2Vec algorithm. Because of the mentioned reasons, LSTM is preferred as the comparison model for the demand prediction step and GraphSAGE is preferred as the comparion model for the product embedding step.

3.4.3.1 Long-short term memory (LSTM)

LSTM is a type of recurrent neural network (RNN) that is able to capture long-term dependencies in sequential data. LSTMs also maintains a memory of past events and then use these past events to make predictions about future events, unlike the traditional RNNs which have limited memory and are unable to capture long-term dependencies. And these benefits make LSTMs useful for time series predictions, where the order of events is important.

An LSTM consists of a series of interconnected cells that process input sequences and maintain a memory of past events. The LSTM may selectively keep or forget knowledge over time thanks to the several gates that regulate the flow of information into and out of each cell. As a result, LSTMs can learn about long-term dependencies in the data and produce predictions that are more precise.

Figure 3.5 shows the LSTM architecture. At each time step t, an LSTM cell takes as input the current input x_t and the previous hidden state h_{t-1}. Using these inputs, LSTM creates outputs; the current hidden state h_t and the current cell state x_t (3.13).

$$h_t, c_t = \text{LSTM}(x_t, h_{t-1}, c_{t-1}) \tag{3.13}$$

The first step in the LSTM is the forget gate (3.15). In this step, it is decided which bits of the cell state are useful according to h_{t-1} and x_t. A neural network (NN) is fed with h_{t-1} and x_t . The NN produces a vector for each piece of information within the range 0 and 1 using the sigmoid activation function (3.14). Indicating that the gate is "closed" and not allowing any information to pass through, a value of 0; indicating that the gate is "open" and allowing information to flow freely, a value of 1.

$$S(t) = \frac{1}{1+e^{-t}} \tag{3.14}$$

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{3.15}$$

The next step is the input gate (3.16). This gate aims to decide what new information should be added to the cell state. Since the function given in equation (3.17) is used as the activation function, the value of new information is between -1 and 1. If the value is positive, the information is added to the cell state, otherwise, it is subtracted. After that, the cell state is updated as given in (3.18).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{3.16}$$

$$N_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{3.17}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{N}_t \tag{3.18}$$

The final step is the output gate (3.19). This gate determines what the cell state should be at time t. Equation (3.19) decides what parts of the cell state we're going to output. Then the cell state is put through $tanh$ and multiplied by o_t (3.20).

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{3.19}$$

$$h_t = o_t * tanh(C_t) \tag{3.20}$$

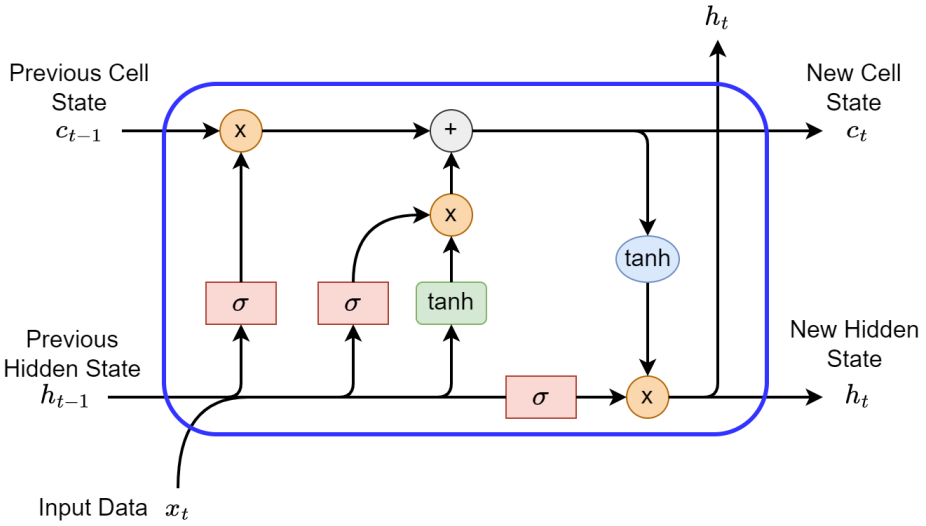


Figure 3.5 : Long-short term memory architecture

3.4.3.2 GraphSAGE

GraphSAGE is defined as a framework for inductive representation learning on large graphs by the authors of GraphSAGE [48]. It is a graph-based machine learning algorithm that uses a combination of aggregation functions and neural networks to generate continuous feature representations for nodes in a graph. In order to find the representation of a node, the algorithm samples a set of neighborhoods of that node. This is accomplished by employing a function that converts the neighborhood into a node embedding and then modifies the representation of the node. With the parameter of K which tells "*how many neighborhoods or how many hops?*" to use to compute the representation of each node, nodes incrementally learn more and more information from their neighborhoods. The same steps are implemented for each node in the graph iteratively and a set of node embeddings are produced that can be applied to later tasks like node classification or link prediction.

Let's assume that GX_i represents the node features for node i, h_i^0 is the initial node embedding representation for node i, h_i^k is the node embedding representation for node

i at the k-th iteration and finally, z_k is the final node embedding representation for a random node i after the GraphSAGE implementation. In the first step, initial node embeddings are set for each node as their feature vectors (3.21).

$$h_i^{k-1} = h_i^0 = x_i \tag{3.21}$$

The next step is the aggregation step (3.22). This step is used to combine the features of the nodes in the neighborhood into a single representation for the target node i. This step aggregates all the embedding vectors for all the nodes j that are in the neighborhood of node i.

$$a_i = f_{aggregate}(h_j | j \in N(i)) \tag{3.22}$$

where $N(i)$ is the set of neighbors of node i. The aggregator functions can be mean aggregator, LSTM aggregator, and pooling aggregator. After calculating the aggregated representation for node i using its neighbors, the representation of node i is updated using its previous representation h_i^{k-1} and aggregated representation a_i (3.23).

$$a_i^k = f_{update}(a_i, h_i^{k-1}) \tag{3.23}$$

3.4.4 Results

Table 3.4 shows the mean absolute error results for XGBR and LSTM. The right side of the table shows the results using the LSTM model and the left side of the table shows the results using XGBR. RandomizedSearchCV is used to tune the hyperparameter of the XGBR model. LSTM is used with 4 units and it is trained for 150 epochs.

In Table 3.4, each row represents MAE results using different graph embeddings, and the results using the graphs are obtained for both XGBR and LSTM. The first row shows the result without using any embedding and we can use this row as a comparison result to see if the graph embeddings improve the prediction results. In the rows, w means with, the number before the character d shows the length of the embedding vector, and the last numerical value shows the hyperparameter value of

the Node2vec which is *length of the random walk*. For the sake of the example, in the second row, $w/10D-100$ means that the row shows the demand prediction results with 10-dimensional embedding obtained when the hyperparameter of Node2Vec *length of the random walk*is equal to 100. The next 5 rows show the embeddings obtained with Node2Vec using the different parameters of *length of the random walk*. Only this parameter is changed in Node2Vec because this parameter determines how close products are visited to create an embedding for a product. And, the last row in the table 3.4 shows the results with graph embeddings obtained using the GraphSAGE algorithm.

When Table 3.4 is examined, the most general comment we can make is that using node embeddings of graphs certainly improves demand forecast results. The embedding 'w/ 10D-80' (10-dimensional, *length of the random walk* = 80) gives the best result using XGBR for Graph 4. It is expected that items bought by the same customer tend to be sold together, and the demand for one product might be similar to another product bought by the same customer. This fact is also supported by the right side of the table. Graph 4 also gives the best result with the embedding 'w/ 10D-GraphSAGE' using LSTM as 2.98.

When GraphSAGE and Node2Vec are compared, it is seen that Node2Vec improves the demand forecasting model more than GraphSAGE. Nevertheless, the best embedding model is GraphSAGE (MAE=2.98) with LSTM as the demand prediction model.According to the results, Node2Vec is chosen as the graph embedding model because it gives better results than the GraphSAGE model. I think the season of the LSTM's poor performance over XGBR is that the dataset has high sparsity and in another saying it has an excessive number of 0 values. Because of the high sparsity of the dataset, LSTM cannot produce results that are superior to those of XGBR. Additionally, Graph 4 sticks out among other graphs for both XGBR and LSTM, demonstrating that employing euclidean similarities to create embeddings makes models more vulnerable.

Table 3.4 : MAE results of XGBR and LSTM with different graph embeddings

	XGBR					LSTM				
	Graph 1	**Graph 2**	**Graph 3**	**Graph 4**	**Graph 5**	**Graph 1**	**Graph 2**	**Graph 3**	**Graph 4**	**Graph 5**
wo/ embedding	2.07	2.07	2.07	2.07	2.07	3.13	3.13	3.13	3.13	3.13
w/ 10D-100	2.06	2.04	2.04	**2.01**	2.03	3.12	3.15	3.11	3.09	3.15
w/ 10D-80	2.05	2.05	2.06	**2.00**	2.06	3.11	3.06	3.10	3.10	3.11
w/ 10D-50	2.06	2.05	2.06	2.05	2.02	3.08	3.13	3.10	3.07	3.15
w/ 10D-20	2.06	2.06	2.06	2.07	2.04	3.12	3.08	3.09	3.06	3.11
w/ 20D-best	2.06	2.04	2.03	2.07	2.02	3.11	3.08	3.14	3.07	3.15
w/ 10D-GraphSAGE	2.07	2.07	2.08	2.05	2.08	3.06	3.05	3.01	**2.98**	3.03

4. TOPIC MODELING ENHANCED TRIPARTITE GRAPH FOR RECOMMENDATION USING METAPATHS

In this chapter, the benefit of topic modeling is used to present a novel graph-based recommendation system technique by creating sub-categories of items with their title information and taking the advantage of the combination of graph embeddings of user-item interactions and topic-aware item representation.

4.1 Background Methods

Two different methods can be explained as background methods which are BERTopic and Metapath2Vec. The BERTopic model is used to learn the item embeddings using their title text and assign each item to the corresponding topic. Then, a heterogeneous tri-partite graph is constructed using user-item and item-topic pairs. Metapath2Vec is used to create item and user embeddings from the heterogeneous tri-partite graph.

4.1.1 Topic modeling with BERT

BERT [49] is a type of language model that uses deep learning to process text written in natural language. BERT stands for "Bidirectional Encoder Representation from Transformers". It extracts the meaning of words in a sentence by considering the surrounding words in the context in which they appear. This allows BERT to understand the nuances and complexities of human language. One of the significant characteristics of BERT is being a bidirectional model, which implies that it considers the context on both the left and right sides of each word in a phrase while processing the sentences. As a result, BERT is able to comprehend the meaning of individual words in a phrase by taking into account the words that follow before and after them. This feature contrasts with many other language models that, while processing the language, often take into account the words that occur before a certain word.

BERTopic [50] uses the BERT model to generate topics from a given document. The model first creates document embeddings using the pre-trained language model

BERT to obtain document-level information. Then, they create clusters of semantically related texts that each represents a different subject by first reducing the dimensionality of document embeddings. They use UMAP as the dimensionality reduction technique because UMAP has no computational restrictions on embedding dimensions. The reduced embeddings are clustered using HDBSCAN which is a variation of the DBSCAN algorithm. HDBSCAN finds clusters by converting DBSCAN into a hierarchical clustering algorithm. This clustering algorithm is chosen because it prevents of assignment of the documents to unrelated clusters. The algorithm labels some documents as outliers if they are not fit into any cluster.

Finally, they use c-TF-IDF to extract topic representations. All the title texts in the same topic are concatenated as a single document. Then, TF-IDF is implemented in each document for each topic (c-TF-IDF). In this way, it is possible o retrieve the top-n words and their corresponding c-TF-IDF score for each topic.

c-TF-IDF, class-based TF-IDF, is a modified version of TF-IDF. The class in the name of the method refers to the topic in BERTopic. To understand the c-TF-IDF better, TF-IDF must be understood first. TF-IDF uses two methods which are Term Frequency (TF) and Inverse Document Frequency (IDF). The TF is the count of words within a document. Equation 4.1 shows the formula of TF for the word a in the document d where N_d^a is the number of occurrences of the word a in the document d and N_d is the total number of words in the document d.

$$TF(a,d) = \frac{N_d^a}{N_d} \tag{4.1}$$

IDF finds how informative certain words are by calculating a word's frequency in a document compared to its frequency across all other documents. Equation 4.2 shows the formula of IDF where N is the total number of documents in the corpus ad n_a is the number of documents in which a appears in d.

$$IDF(a) = \frac{N}{n_a} \tag{4.2}$$

Then, TF-IDF produced a sparse word matrix that can be used for many implementations such as document similarity and feature extraction. In c-TF-IDF, the class-based TF-IDF, each document is associated with one or more predefined classes (topics).

The formula of c-TF-IDF is given in Equation 4.3 where c is the class (topic) and it uses a different version of IDF as given in Equation 4.4 where N_c is the total number of documents in class c and n_c is the number of documents in class c that contains the word a.

$$c-TF-IDF(a,d,c) = TF(a,d) \times IDF(a,c) \tag{4.3}$$

$$IDF(a,c) = \frac{N_c}{n_c} \tag{4.4}$$

4.1.2 Metapath2Vec

The Metapath2Vec model uses a heterogeneous skip-gram model to construct node embeddings by formalizing metapath based random walks. It uses the node's heterogeneous neighborhood. One of the main contributions of the model over the other node embedding models, it can work with heterogeneous networks and learn the low-dimensional embeddings for multiple types of nodes. As the first step, the algorithm implements metapath based random walks in the heterogeneous network. Then, they use an extended version of the skip-gram model to make it easier to describe nodes that are near in both space and semantics.

As the first step, the algorithm uses a uniform random walk to generate a list of node IDs from G. It assumes the list of node IDs as sentences and the set of all sentences as a corpus. After that, each node ID is considered as different words in the dictionary and then, Word2Vec algorithm [45] is used to calculate the embeddings $\underset{u}{\rightarrow}$ and $\underset{i}{\rightarrow}$.

Given the heterogeneous network G and the metapath schema MP, the goal is learning the d-dimensional latent representations $X \in \mathbb{R}^{|V| \times d}$. The metapath schema defines the path that the model should follow. For example, in a heterogeneous graph, there are 3 different node types A, B, C and all the node types can connect to each other. In this

graph, one of the metapath schemas we can define is $MP = \{(A \rightarrow B \rightarrow A), (B \rightarrow C \rightarrow A \rightarrow C \rightarrow B)\}$. It means that the model should follow that scheme when implementing walks.

4.2 Proposed Model

In this section, the methodology is explained step by step. First, the topic modeling part using BERT, and then the information about the tri-partite graph is explained. Finally, the last part gives the details of the proposed model.

Problem Definiton Given a set of users U, set of items I and set of topics T, the aim is to build vector-space representation for each user $u \in U$ and for each item $i \in I$, and then recommend $top-K$ items to each user. If there are N_u number of users, N_i number of items, and N_t number of topics, there are $N_u + N_i + N_t$, number of nodes in the graph.

Assume that there is a heterogeneous graph $G = (V, E, F, R, \phi, \varphi)$ where F is the set of node types and R is the set of edge types. Each node $v \in V$ has associated with a node type function $\phi(v) : V \rightarrow F$ and each edge $e \in E$ is associated with an edge type mapping function $\varphi : E \rightarrow R$, where $|F| + |R| > 2$. As shown in Fig. 4.3, a heterogeneous graph is constructed with three types of nodes (user, item, topic) and two types of edges (user-item, item-topic). Given the heterogeneous graph G, the goal of the model is to build low-dimensional representations $\underset{u}{\rightarrow}$ and $\underset{i}{\rightarrow}$ for each $u \in U$ and for each $i \in I$.

4.2.1 Creating topic nodes

BERTopic is used to assign a topic to each item based on its title. The topic assigned to each item corresponds to the sub-category of that item. In this way, it is possible to categorize items even if there is no category information of the items in a dataset. In this study, obtained topics are called sub-categories since each data set used in the experiments consists of items of one category (Amazon Beauty and Amazon Video Games). However, if the items in the data set are from different categories, the BERTopic model can be used again to obtain topics as categories.

At this step, the topic modeling approach is improved. If the HDBSCAN model can't assign a document (item) to any cluster (topic), a new cluster named -1 is created and includes all the uncategorized documents (items) [50].

To prevent the uncategorized items, the BERTopic model is enriched so that it calculates the closest topic for the item marked as an uncategorized one and assigns the item to the closest topic. In this way, it is possible to assign a topic for each item without exception. The reason why it is aimed to prevent the uncategorized item is explained in subsection 4.3.

4.2.2 Graph embedding with topic nodes

Tripartite Graph. Each user, item, and topic is represented as a node in the heterogeneous tripartite undirected graph. If a user u interacts with an item i, then an edge is constructed between them. There is an edge between item i and topic t, if item i belongs to topic t. Thus, the undirected heterogeneous graph G has three different node types: u, i, and t. There is a many-to-many relation between items and users, one user can buy more than one item and one item can be bought by more than one user, while one item can be only in one topic.

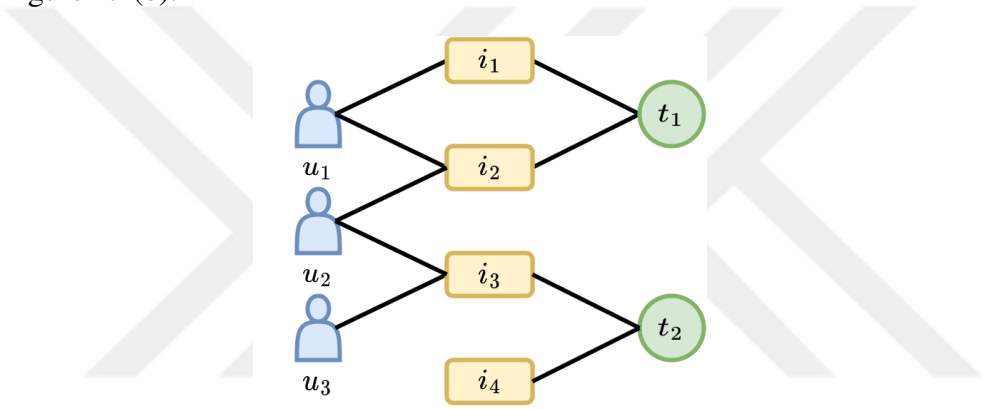
The graph G is constructed using the interactions between users and items and the interactions between items and topics. For the interaction between users and items, only the train set is used, the interactions from the test set are excluded.

The algorithm used to learn the low-dimensional embeddings for multiple types of nodes in a heterogeneous network is Metapath2Vec. It performs random walks on G by creating metapaths. Using its metapath advantages, different metapaths can be defined and learned accurate embeddings according to the problem.

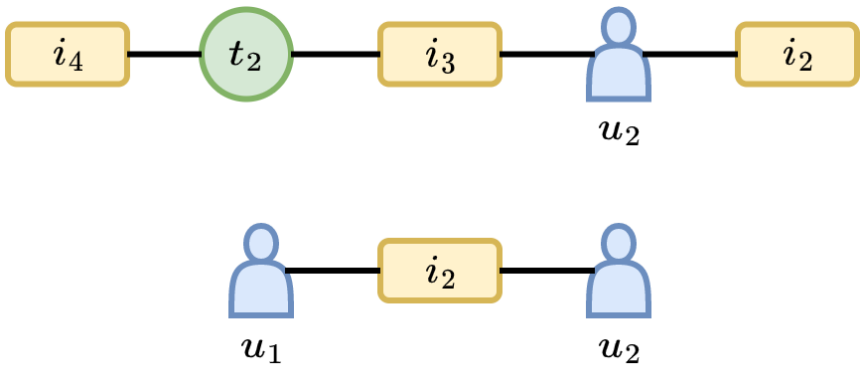
Given the heterogeneous network G and the metapath schema MP, the goal is learning the d-dimensional latent representations $X \in \mathbb{R}^{|V| \times d}$. In the implementation, the metapath schema is $MP = \{(u \rightarrow i \rightarrow u), (i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)\}$. In the case that the metapath $(u \rightarrow i \rightarrow u)$ doesn't work when the item is a sparse item, in other words, if the item interacts with very few users, the metapath $(i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)$ allows us to obtain more accurate embeddings about them. There is no such situation in this dataset,

but even if some products are not interacting with any users, we can get information thanks to our metapath that includes the topic node for those products. When the path starts from a sparse item, there is no possible path toward any user, as a result, it is not possible to learn transactional information about that item using $(u \rightarrow i \rightarrow u)$ path, instead, the transactional information of the similar item (within the same topic) can be used to learn more accurate embeddings.

The heterogeneous graph can be seen in Figure 4.1(a). If a metapath $(u \rightarrow i \rightarrow u)$ starts from any user, the path can't reach i_4 from the users because i_4 is a cold-start item with zero user relation in the graph. Thanks to the proposed approach, the 2-hop user node neighbor can be reached via i_3 by following the first metapath $(i_4 \rightarrow t_2 \rightarrow i_3 \rightarrow u_2 \rightarrow i_2)$ from Figure 4.1(b).



(a) Heterogeneous tri-partite graph



(b) Metapaths

Figure 4.1 : Heterogeneous tri-partite graph and an example implementation metapath schema *MP*

Neural Network Architecture. This part, it is aimed to further develop product embeddings by observing the relations of products with users by using a simple neural network. The neural network consists of an input layer, embedding layer, dot layer,

and output layer as shown in Figure 4.2. Figure only shows the architecture for u_1 and i_1. Users and items are used as the input layer and they are given as the embeddings to the model. The initial item embeddings and user embeddings in the neural network are the graph embeddings obtained using Metapath2Vec. The Embedding Layer is updated during the learning phase. The Dot Layer merges the Embedding Layer by using the dot product. The model is trained by trying to predict the likelihood that the product and user will interact. Negative sampling is used to create the training set for the neural network. The user-item relations that are not included in the dataset were created randomly and these relations were used as negative samples. The user-item relations in the dataset were used as positive samples.

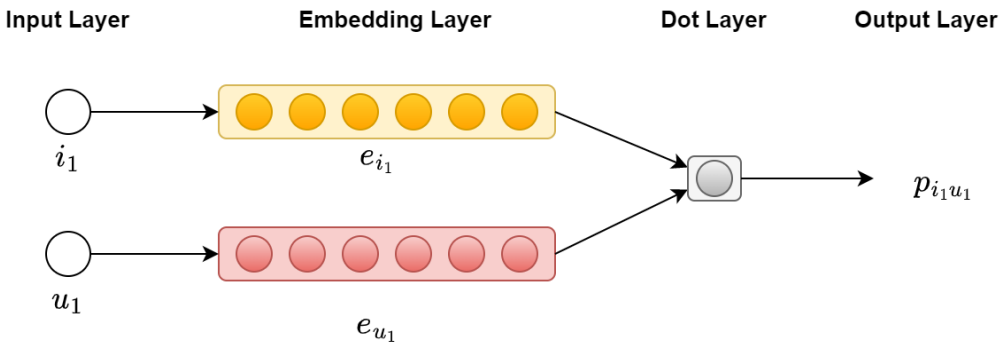


Figure 4.2 : Neural network architecture

Obtaining Recommendations. After obtaining the final embedding using Neural Network architecture, the inner product of the item and user embeddings is taken to find a score. Each user-item pair (u,v) gets a score showing the relationship between them. A higher score corresponds to a stronger relationship between the user u and the item i meaning that user u will probably prefer the item i. The $top-M$ items are recommended with the highest scores to the users.

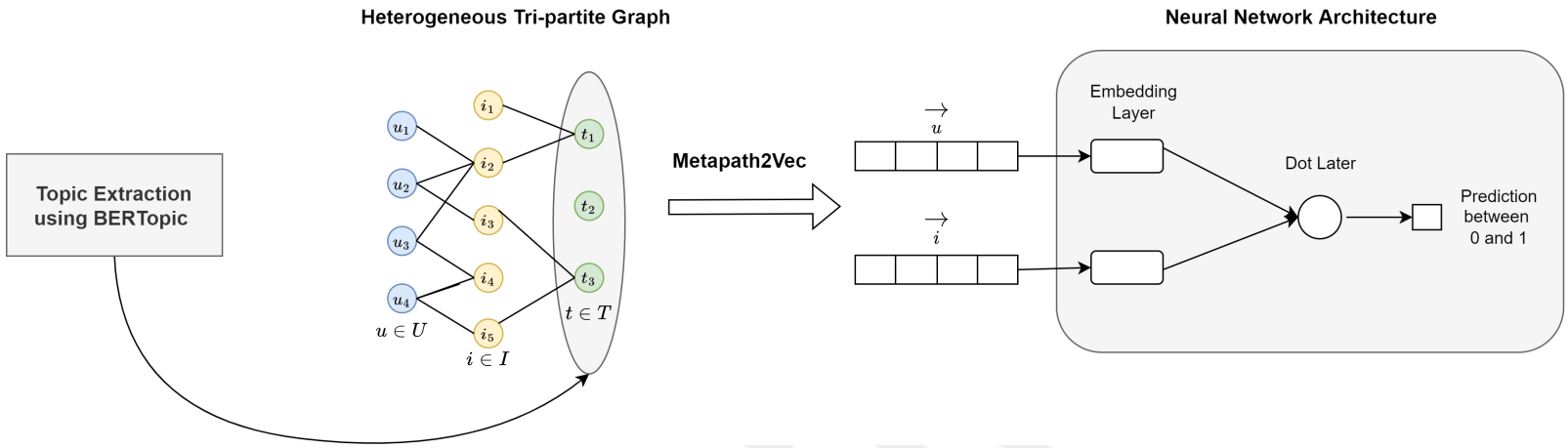


Figure 4.3 : General workflow of the proposed model

4.3 Dataset

Two different amazon datasets are used which are from the categories Amazon All Beauty and Amazon Video Games [1]. The Amazon datasets contain information about customer reviews and product-meta data such as product titles, and product descriptions. These datasets do not include data on whether customers have bought the products. But in this study, the graphs are constructed based on the purchasing relationship between customers and products. As a result, if a customer reviews a product, it means that the user is interested to buy in the product. These datasets are not used for Chapter 3, because there is no invoice information in Amazon datasets.

Because of the high sparsity of the dataset, only the products and items are filtered that have relationships greater than a specific threshold. For Amazon All Beauty dataset, the threshold value is chosen as 5, while it is chosen as 12 for the Amazon Video Games dataset. The threshold value 5 is chosen because this value is chosen in the dataset link to create more dense subsets of the dataset. The data are reduced to extract the k-core, such that each of the remaining users and items has k number of relationships each. The threshold value is increased to 12 for the Video Games dataset because this dataset is big and we want to reduce the dataset as much as possible.

In order to create embeddings from the title of products and use them to assign topics, some data preparation steps are applied. During the data preparation step, all the stop words, digits, and punctuation are removed from the titles of products, and all characters are converted to lowercase.

After that, the customer's interactions are sorted by time and the most recent interaction of each customer is separated for the test set. All the other interactions but the last one are used for the training set.

Table 4.1 shows the statistical information of Amazon All Beauty and Amazon Video Games. All Beauty dataset is smaller and has fewer customers and products than the Video Games dataset.

[1] https://nijianmo.github.io/amazon/index.html, December 2022

Table 4.1 : Dataset information

Dataset	#Customers	#Products	#Topics	#Interactions	Density
Beauty	1.488	1.395	22	6.961	0,00335
Video Games	19,683	9,349	447	201,869	0.00109

How to handle uncategorized item?

If the topic modeling approach marks an item as uncategorized, in other words, if it cannot assign the item to any topic, that item is assigned to the closest topic. Assign a topic for every item without exception is important because the -1 topic contains very unrelated items together and it may cause to get incorrect information for these items when using the metapath $(i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)$. Since the goal is to learn information from the items on the same topic, assigning all the items to the most relevant topics is essential to obtain the correct information.

At this stage, firstly a BERT embedding is created for each topic using n number of words with the BERT model. The n number of words is given by the BERTopic model and they are the most frequent words for the topic. Then, the sum of BERT embeddings of n number of words is taken and divided by n to find the topic embeddings. Then, for each uncategorized item, item embeddings are created with the BERT model using the title information of the item as implemented in this study [51]. After that, the cosine similarity between the item embeddings and each topic embedding is calculated. The closest topic for each uncategorized item is accepted as the topic for that item.

4.4 Experiments

Parameter Settings. StellarGraph [47] library is used for graph construction and Metapath2Vec implementation. Keras [52] library is used for the NN implementation. To generate BERT embedding from an item title, HuggingFace BERT implementation [53] is used. In the Metapath2Vec algorithm, the walk length is set to 20, and the number of random walks per node is set to 5. While training the NN, the batch size is set to 32, epoch to 200, and negative ratio to 2.

4.4.1 Evaluation metrics

In order to evaluate the success of the proposed model it is preferred to use the most popular ranking metrics *Hit Ratio (HR)* and *Normalized Discounted Cumulative Gain (NDCG)*. These metrics allow measuring the ranking quality of our top-K recommendation. K is set as 20 and the average metrics are reported for all users in the test set [35].

- **Hit Ratio**: It is the metric of counting the number of hits in a k-sized list of rated items without considering the order of the recommendation. It is calculated by dividing the number of hits by the number of recommendations.

$$HR@k = \frac{\#of hits}{k\, number of recommendations} \tag{4.5}$$

- **Normalized Discounted Cumulative Gain (NDCG):** It is important to understand the Cumulative Gain (CG) and Discounted Cumulative Gain (DCG) in order to understand the formula of NDCG. CG is the sum of all the relevance scores in the recommendation set (4.6) where $relevance_i$ is the ranking score at position i.

$$CG = \sum_{i=1}^{n} relevance_i \tag{4.6}$$

  Because CG doesn't consider the position of the relevant items, different recommendation lists might look the same even though one of the lists is much better than the other list according to the position of the relevant items. Equation (4.7) shows the calculation of DCG by dividing the relevance score by the log of the corresponding position where p is the particular rank position.

$$DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{4.7}$$

  NDCG is more useful than DCG to compare the performances of different models that result in different recommendation results lists because NDCG normalizes the cumulative gain at each position for a specified item across different models.

Equation (4.8) shows the formula of NDCG where $|REL|$ is the list of documents ordered by relevance in the corpus up to position p.

$$nDCG_p = \frac{DCG_p}{IDCG_p} \tag{4.8}$$

where

$$IDCG_p = \sum_{i=1}^{|REL_p|} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{4.9}$$

4.4.2 Comparison models

The following models are used as the baseline models and the hyperparameters are set as in Polignano et. al.'s study [37] using NeuRec library [2]:

- **Matrix Factorization (MF) [20]:** The method uses the collaborative filtering-based approach and aims to find the relationship between users and items. MF is trying to estimate a relation $\hat{y}_{ui}$. It is calculated as a simple dot product of p_u and q_i where p_u and q_i are the latent vectors for user u and item i and K is the dimension of the latent space as shown in Equation 4.10.

$$\hat{y}_{ui} = f(u,i|p_u,q_i) = p_u^T q_i = \sum_{k=1}^{K} p_{uk} q_{ik} \tag{4.10}$$

- **Multi-Layer Perceptron (MLP) [19] :** It is an item-based collaborative filtering system. It aims to capture the non-linear relationship between user u and item i and performs better than MF. It can also capture the high-order interactions between users and items as shown in this study [19]. MLP is a type of feedforward artificial neural network that consists of multiple layers of perceptrons, which are simple units that compute a linear combination of their inputs and pass the result through a non-linear activation function. It is trained by trying to predict ratings for the user-item pairs.

- **Neural Matrix Factorization (NeuMF) [19]:** It is an item-based collaborative filtering based model. It benefits from the combination of MF and MLP models.

[2] https://github.com/wubinzzu/NeuRec

NeuMF consists of 4 main layers which are the input layer, embedding layer, neural CF layers, and output layer. The input layer binarizes the one-hot encoded vectors for a user and item identification where 1 means the user u interacted with item i. The embedding layer is a fully connected layer that converts the one-hot encoded vectors to dense vectors. The obtained dense vectors are the user and item vectors (latent user and item vectors). The next layers are the Neural CF layers. They use multi-layered neural architecture to map the embeddings to prediction scores. Finally, the output layer returns the predicted score by minimizing the pairwise loss.

- **Denoising Auto-Encoder (DAE): [54]** It is a type of autoencoder, which is a neural network architecture. An autoencoder consists of two main components: an encoder and a decoder. The encoder maps the input data to a lower-dimensional representation, called the bottleneck or latent representation. The decoder then maps the bottleneck representation back to the original input space. The DAE is a modified version of AEs. In the corruption process, some of the interactions are masked randomly. Then, DAE is trained by learning a robust representation of the users and items that is able to reconstruct the original interactions despite the added noise. The representations learned by the DAE are used as the user and item embeddings in a recommendation system.

- **Neural Graph Collaborative Filtering (NGCF) [35]:** In this study, the authors propose a recommendation system by exploiting the user-item graph. By taking benefit from the high-order connectivity of the user-item graph, they propagate embeddings and improve the recommendation system using a bipartite graph. NGCF uses an encoder-decoder architecture to model the user-item interactions, the encoder is used to learn the representations of the users and items, while the decoder is used to predict the ratings. The encoder is typically implemented as a GNN and the decoder is typically implemented as a neural network. The model is trained to minimize the difference between the predicted ratings and the true ratings.

4.4.3 Results

The experiments are implemented from three different perspectives. Firstly, the model is compared with the baselines, and a special scenario which is cold-start is implemented. Then, the results of different graph embeddings are compared. As the final experiment, visualizations are used to show some case studies.

4.4.3.1 Comparison the baselines & cold-start scenario

The proposed model is compared with the baselines in two cases in the first experimental group. Besides showing the performance of the proposed model in the general test set, results with the small sparse test set are given to present that the proposed model can make better predictions even for sparse items. While Table 4.2 shows the results of baseline models and our result using the entire test set, Table 4.3 shows the results using only the small test set with sparse items.

According to Table 4.2, the proposed model is being outstanding to all baselines in the HR metric by at least 3.8 %. The model only outperforms the Video Games dataset in the NDCG metric. It might be caused by the size difference between the two datasets because the Video Games dataset is bigger in size compared to the Beauty dataset. Another reason is the diversity of the items in the datasets. The items in the Beauty dataset repeat themselves and there is little variety in items. In the Beauty dataset, most of the products sold are repeating themselves. According to the data analysis, only the 7 products make up almost 50% of total sales. 2874 of 5473 sales include sales of only 7 products. Nevertheless, in the Video Games category, there are no products that take up most of the sales. The products generally have similar volumes of sales.

To investigate how the model performs for the items that have a very low number of interactions, a small test dataset is created by selecting the items from the test set that have only 1 or 2 interactions in the train set. Table 4.3 shows the metric results of the baselines and our model with the sparse test set. The results from Table 4.3 show that the model makes better predictions than the baselines for the sparse dataset. The reason why the baseline models can't predict a newly added item with only a

Table 4.2 : Baseline results with entire test set

	Amazon Beauty		Amazon Video Games	
	HR@20	NDCG@20	HR@20	NDCG@20
MLP	0.7484	0.5048	0.0352	0.0130
MF	0.8109	0.6938	0.0471	0.0192
NeuMF	0.6999	0.5737	0.0485	0.0185
DAE	0.6773	0.5948	0.0186	0.0079
NGCF	0.8091	0.6997	0.0581	0.0220
Our Model	**0.8421**	0.4781	**0.0837**	**0.0365**

few interactions successfully that they use the CF-based approach or user-item based graph structure. On the other hand, a connection in sparse items by only looking at the relationship between items and users wasn't established. A hybrid model is developed that could include the relationships of the items fed from the item's title information in the modeling of the user-item relationship, and it allowed us to achieve more successful results.

Table 4.3 : Baseline results with sparse test set

	Amazon Beauty		Amazon Video Games	
	HR@20	NDCG@20	HR@20	NDCG@20
MLP	0.0603	0.0324	0.0033	0.0018
MF	0.0862	0.0403	0.0008	0.0002
NeuMF	0.0948	0.0683	0.0067	0.0021
DAE	0.0258	0.0107	0.0000	0.0000
NGCF	0.0689	0.0343	0.0016	0.0003
Our Model	0.1034	0.0369	0.0504	0.0182

Although the model couldn't pass all of the baseline results according to the NDCG metric using the sparse dataset for the Beauty dataset, it was able to pass 2 baseline results and gave results close to the other two baselines compared to the entire test set. Again, the model gives better results than the baselines according to HR for both datasets.

Table 4.2 shows how the model outperforms the baselines thanks to the topic node used while building our model, and the metapath $(i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)$ used to get more information about sparse items.

4.4.3.2 Comparison of the different graph embeddings

The second experiment compares the proposed model with the different graph embeddings. Besides the tri-partite graph, a bi-partite graph is also created by using only user-item interaction. Only two different node types U and I are used, and there are edges only between items and users. Table 4.4 shows the comparison between different graph embeddings. The proposed model in the last row outperforms all results. Metapath2Vec algorithm with the tri-partite graph gives the lowest score. After these embeddings are trained using NN, the best score is obtained in our proposed model. Besides the proposed model, the second and third-best scores are obtained using the Metapath2Vec algorithm with a bipartite graph and the Node2Vec algorithm with a bipartite graph. As expected, NN improves the embeddings obtained from tripartite Metapath2Vec, because it can learn relationships between input variables and output variables better rather than our graph representation since just $(u \rightarrow i \rightarrow u)$ metapaths aren't used, $(i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)$ metapath is also used.

Probably it is possible to learn the user-product relationships more accurately by using NN when a more complex metapath is used, and the results support this. Especially, to demonstrate the outstanding contribution of the $(i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)$ metapath, the result of the model is compared with the result of the model without using that metapath. The last two rows from Table 4.4 shows the results. While the first model shows the result of the model without using that metapath, the second model also includes $(i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)$ metapath. The results show that the usage of that metapath improves HR by 3.5% and NDCG by 21% for the Beauty dataset and improves HR by 161% and NDCG by 165% for Video Games dataset compared to the result without using $(i \rightarrow t \rightarrow i \rightarrow u \rightarrow i)$ metapath.

4.4.3.3 Case study

Besides the evaluation part using metrics, there are also some visualizations for example topics and an example for the assignment of the uncategorized items to the closest topics.

Table 4.4 : Comparison of different graph embeddings

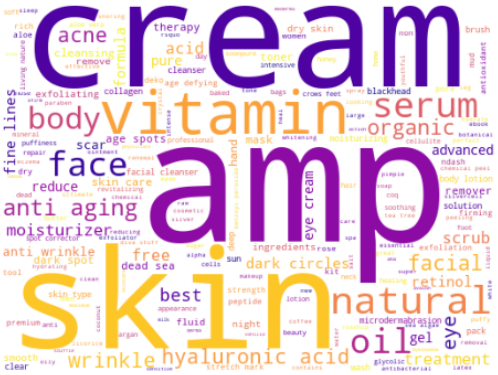
	Amazon Beauty		Amazon Video Games	
	HR@20	NDCG@20	HR@20	NDCG@20
Bi-HinSAGE	0.6018	0.3556	0.0775	0.0305
Bi-Node2Vec	0.8125	0.4774	0.0394	0.0132
Tri-Metapath2Vec	0.7840	0.4357	0.0254	0.0091
Bi-Metapath2Vec	0.8197	0.4724	0.0016	0.0006
Bi + NN	0.8126	0.3841	0.0090	0.0030
Tri + NN	**0.8421**	**0.4781**	**0.0837**	**0.0365**

Topic Word Visualization: To show the success of our topic modeling, example word cloud visualizations are given that show the most common words in the topics. In the word cloud, words are written bigger the more often they are mentioned. Figure 4.4(a) and 4.4(b) show the two most dense topics from Beauty Dataset. It is clear that one topic includes items about nails, nail art, and other nail products such as polish, stickers, manicure, while another topic includes items about skincare, cream, anti-aging products, and serums. The word *amp* is a skincare brand. Figure 4.4(c) and 4.4(d) give the most common words from Video Games dataset topics. Figure 4.4(c) includes items about Nintendo DS and its games. Figure 4.4(d) shows that it includes items about Xbox.

Assigning uncategorized items to the closest topics: In this part, the assignment quality of the uncategorized items to the closest topics is shown. Random items from uncategorized topics are chosen and they are investigated according to the compatibility of the assignments. Figure 4.4(b) shows the Beauty Skin Care / Cream topic and randomly three uncategorized items that are assigned to that topic. The titles of the items with the keywords (most frequent words in the topic) in bold are as follows: "***aloe*** *vera gel huge* ***organic*** *cold pressed aloe nutrilab naturals* ***natural*** *ingredients use eczema sun burn sun bug insect bites aloe rich* ***vitamins*** *e c b vitamins nourish* ***skin*** *aloe vera grown bottled usa* ***amp*** *certified organic texas department agriculture*" (1), "*natures greatest secret amber formula antibacterial antifungal colloidal silver coconut* ***oil*** *intensive formula soothing* ***skin*** *treatment* (2)", "*alexa* ***organic*** *kona coffee* ***scrub dead sea*** *salt organic olive* ***oil*** *sweet almond oil grape seed oil shea butter* ***amp***

(3)". If you compare the most frequent words you can see that there are many common words that prove the success of the uncategorized item assignment approach.

(a) Beauty-Nail / Nail Art



(b) Beauty-Skin Care / Cream

(c) Video Games-Nintendo DS

(d) Video Games-Xbox

Figure 4.4 : Word Cloud Visualization

5. CONCLUSION AND FUTURE WORKS

This thesis proposes two different novel frameworks for demand prediction and recommendation system by taking the advantage of graph embeddings.

The first implementation aims to enrich the demand forecasting model by using the different relationships of products. Different relationships between products are constructed with graph structure using the transactional sales data of products, and graph embedding techniques are implemented to learn the relationships between the products. To implement the proposed model, firstly the Apriori algorithm and K-Means Clustering are used to extract the relationship between products. The next step is extracting the vector representation of products. The Node2Vec algorithm and GraphSAGE algorithm are implemented, and according to the MAE metric, the more successful algorithm becomes the Node2Vec algorithm. Finally, the vector representations or the embeddings of products are used with the previous sales data to predict the demand value of the products. XGBR and LSTM models are used to make time-series predictions. According to the results, Graph 4, which represents the relationship between products using euclidean distance, gives the best result as MAE=2.00 with the hyper-parameters of the Node2Vec where the embedding size is 10 and the walk length is 80 using XGBR. LSTM also gives the best result using Graph 4 with 10D embedding as MAE=2.98. When XGBR and LSTM are compared, XGBR gives better results according to the MAE. However, the results show that the usage of embeddings improved the demand prediction for both XGBR and LSTM.

The second implementation proposed a hybrid recommendation system by using topic-enriched item embeddings. The model is improved with topic modeling so that it is possible o extract meaningful embeddings even for sparse items. As the first step, topics are extracted for each item using title information. At this point, the topic modeling is enriched by assigning uncategorized items to the closest topics. After that, a heterogeneous tripartite graph is constructed and the Metapath2Vec algorithm

is implemented with reasonable metapaths. The evaluated embeddings are used as the initial embedding for NN, and finally, the recommendations are implemented. Various experiments on two different Amazon datasets show that the proposed model can predict effective recommendations. Besides, the case studies demonstrate the success of the approach to topic modeling.

The future works can be explained from three different perspectives which are the demand prediction model, the recommendation system, and the future works for the general of the thesis. In the demand prediction model, more than one dataset should be used to evaluate the success of the approach. It also helps to understand the usability of the model in different fields, where there are transactional data, not only in the retail industry. For the recommendation system, the model can be evaluated using one more dataset and the case studies can be extended to add more explainability to the implementation. In addition, the users can be grouped as it is implemented for items as topics. If the reviews made by users to items are added to the model, more meaningful embeddings can be obtained. Finally, as the future works for all the parts of the thesis, the usage of different graph neural network models helps us to observe better results.

REFERENCES

[1] **Yilmaz, Y. and ÖĞüdücü, c.G.** (2022). Enriching Demand Prediction with Product Relationship Information Using Graph Neural Networks, *Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, ASONAM '21, Association for Computing Machinery, New York, NY, USA, p.561–568, https://doi.org/10.1145/3487351.3489477.

[2] **Wu, Z.**, **Pan, S.**, **Chen, F.**, **Long, G.**, **Zhang, C. and Yu, P.S.** (2021). A Comprehensive Survey on Graph Neural Networks, *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 4–24.

[3] **Perozzi, B.**, **Al-Rfou, R. and Skiena, S.** (2014). Deepwalk: Online learning of social representations, *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp.701–710.

[4] **Zhou, J.**, **Cui, G.**, **Hu, S.**, **Zhang, Z.**, **Yang, C.**, **Liu, Z.**, **Wang, L.**, **Li, C. and Sun, M.** (2020). Graph neural networks: A review of methods and applications, *AI Open*, *1*, 57–81.

[5] **Li, S.**, **Kawale, J. and Fu, Y.** (2015). Deep collaborative filtering via marginalized denoising auto-encoder, *Proceedings of the 24th ACM international on conference on information and knowledge management*, pp.811–820.

[6] **Sunil Chopra, P.M.** (2017). *Supply chain management: Strategy, planning, and operation*.

[7] **Ferbar, L.**, **Čreslovnik, D.**, **Mojškerc, B. and Rajgelj, M.** (2009). Demand forecasting methods in a supply chain: Smoothing and denoising, *International Journal of Production Economics*, *118*(1), 49–54, https://www.sciencedirect.com/science/article/pii/S0925527308002466, special Section on Problems and models of inventories selected papers of the fourteenth International symposium on inventories.

[8] **Harrison, P.** (1967). Exponential smoothing and short-term sales forecasting, *Management Science*, *13*(11), 821–842.

[9] **Fattah, J.**, **Ezzine, L.**, **Aman, Z.**, **El Moussami, H. and Lachhab, A.** (2018). Forecasting of demand using ARIMA model, *International Journal of Engineering Business Management*, *10*, 1847979018808673.

[10] **Ramos, P.**, **Santos, N. and Rebelo, R.** (2015). Performance of state space and ARIMA models for consumer retail sales forecasting, *Robotics and computer-integrated manufacturing*, *34*, 151–163.

[11] **Falatouri, T.**, **Darbanian, F.**, **Brandtner, P. and Udokwu, C.** (2022). Predictive Analytics for Demand Forecasting–A Comparison of SARIMA and LSTM in Retail SCM, *Procedia Computer Science*, *200*, 993–1003.

[12] **Elmasdotter, A. and Nyströmer, C.** (2018). *A comparative study between LSTM and ARIMA for sales forecasting in retail*.

[13] **Jain, A.**, **Karthikeyan, V.**, **Sahana, B.**, **Shambhavi, B.**, **Sindhu, K. and Balaji, S.** (2020). Demand Forecasting for E-Commerce Platforms, *2020 IEEE International Conference for Innovation in Technology (INOCON)*, IEEE, pp.1–4.

[14] **Kilimci, Z.H.**, **Akyuz, A.O.**, **Uysal, M.**, **Akyokus, S.**, **Uysal, M.O.**, **Atak Bulbul, B. and Ekmis, M.A.** (2019). An improved demand forecasting model using deep learning approach and proposed decision integration strategy for supply chain, *Complexity*, *2019*.

[15] **Benhamida, F.Z.**, **Kaddouri, O.**, **Ouhrouche, T.**, **Benaichouche, M.**, **Casado-Mansilla, D. and López-de Ipiña, D.** (2020). Stock&Buy: A New Demand Forecasting Tool For Inventory Control, *2020 5th International Conference on Smart and Sustainable Technologies (SpliTech)*, IEEE, pp.1–6.

[16] **Qi, Y.**, **Li, C.**, **Deng, H.**, **Cai, M.**, **Qi, Y. and Deng, Y.** (2019). A deep neural framework for sales forecasting in e-commerce, *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pp.299–308.

[17] **Shi, J.**, **Yao, H.**, **Wu, X.**, **Li, T.**, **Lin, Z.**, **Wang, T. and Zhao, B.** (2021). Relation-aware Meta-learning for E-commerce Market Segment Demand Prediction with Limited Records, *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, pp.220–228.

[18] **Liao, S.**, **Yin, J. and Rao, W.** (2020). Towards Accurate Retail Demand Forecasting Using Deep Neural Networks, *International Conference on Database Systems for Advanced Applications*, Springer, pp.711–723.

[19] **He, X.**, **Liao, L.**, **Zhang, H.**, **Nie, L.**, **Hu, X. and Chua, T.** (2017). Neural Collaborative Filtering, *CoRR*, *abs/1708.05031*, `http://arxiv.org/abs/1708.05031`, `1708.05031`.

[20] **Koren, Y.**, **Bell, R. and Volinsky, C.** (2009). Matrix factorization techniques for recommender systems, *Computer*, *42*(8), 30–37.

[21] **Lee, D. and Seung, H.S.** (2000). Algorithms for non-negative matrix factorization, *Advances in neural information processing systems*, *13*.

[22] **Sedhain, S.**, **Menon, A.K.**, **Sanner, S. and Xie, L.** (2015). Autorec: Autoencoders meet collaborative filtering, *Proceedings of the 24th international conference on World Wide Web*, pp.111–112.

[23] **Shu, J.**, **Shen, X.**, **Liu, H.**, **Yi, B. and Zhang, Z.** (2018). A content-based recommendation algorithm for learning resources, *Multimedia Systems*, *24*(2), 163–173.

[24] **Bouihi, B. and Bahaj, M.** (2019). Ontology and Rule-Based Recommender System for E-learning Applications., *International Journal of Emerging Technologies in Learning*, *14*(15).

[25] **Walek, B. and Fajmon, P.** (2023). A hybrid recommender system for an online store using a fuzzy expert system, *Expert Systems with Applications*, *212*, 118565.

[26] **Vahidi Farashah, M.**, **Etebarian, A.**, **Azmi, R. and Ebrahimzadeh Dastjerdi, R.** (2021). A hybrid recommender system based-on link prediction for movie baskets analysis, *Journal of Big Data*, *8*(1), 1–24.

[27] **Wang, H.C.**, **Jhou, H.T. and Tsai, Y.S.** (2021). Adapting topic map and social influence to the personalized hybrid recommender system, *Information Sciences*, *575*, 762–778.

[28] **Schafer, J.B.**, **Frankowski, D.**, **Herlocker, J. and Sen, S.** (2007). Collaborative Filtering Recommender Systems, Springer-Verlag, Berlin, Heidelberg, p.291–324.

[29] **Berg, R.v.d.**, **Kipf, T.N. and Welling, M.** (2017). *Graph Convolutional Matrix Completion*, `https://arxiv.org/abs/1706.02263`.

[30] **Chen, L.**, **Wu, L.**, **Hong, R.**, **Zhang, K. and Wang, M.** (2020). *Revisiting Graph based Collaborative Filtering: A Linear Residual Graph Convolutional Network Approach*, `https://arxiv.org/abs/2001.10167`.

[31] **He, X.**, **Deng, K.**, **Wang, X.**, **Li, Y.**, **Zhang, Y. and Wang, M.** (2020). Lightgcn: Simplifying and powering graph convolution network for recommendation, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp.639–648.

[32] **Wang, F.**, **Li, Y.**, **Zhang, Y. and Wei, D.** (2022). KLGCN: Knowledge graph-aware Light Graph Convolutional Network for recommender systems, *Expert Systems with Applications*, *195*, 116513, `https://www.sciencedirect.com/science/article/pii/S0957417422000148`.

[33] **Wu, C.**, **Liu, S.**, **Zeng, Z.**, **Chen, M.**, **Alhudhaif, A.**, **Tang, X.**, **Alenezi, F.**, **Alnaim, N. and Peng, X.** (2022). Knowledge Graph-Based Multi-Context-Aware Recommendation Algorithm, *Inf. Sci.*, *595*(C), 179–194, `https://doi.org/10.1016/j.ins.2022.02.054`.

[34] **Wang, P.**, **Li, X.**, **Du, F.**, **Liu, H. and Zhi, S.** (2019). A Personalized Recommendation System based on Knowledge Graph Embedding and Neural Network, *2019 3rd International Conference on Data Science and Business Analytics (ICDSBA)*, pp.161–165.

[35] **Wang, X.**, **He, X.**, **Wang, M.**, **Feng, F. and Chua, T.S.** (2019). Neural Graph Collaborative Filtering, *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, Association for Computing Machinery, New York, NY, USA, p.165–174, https://doi.org/10.1145/3331184.3331267.

[36] **Li, Z.**, **Shen, X.**, **Jiao, Y.**, **Pan, X.**, **Zou, P.**, **Meng, X.**, **Yao, C. and Bu, J.** (2020). Hierarchical Bipartite Graph Neural Networks: Towards Large-Scale E-commerce Applications, *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, IEEE, pp.1677–1688.

[37] **Polignano, M.**, **Musto, C.**, **de Gemmis, M.**, **Lops, P. and Semeraro, G.** (2021). Together is Better: Hybrid Recommendations Combining Graph Embeddings and Contextualized Word Representations, Association for Computing Machinery, New York, NY, USA, p.187–198, https://doi.org/10.1145/3460231.3474272.

[38] **Fan, S.**, **Zhu, J.**, **Han, X.**, **Shi, C.**, **Hu, L.**, **Ma, B. and Li, Y.** (2019). Metapath-Guided Heterogeneous Graph Neural Network for Intent Recommendation, *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '19, Association for Computing Machinery, New York, NY, USA, p.2478–2486, https://doi.org/10.1145/3292500.3330673.

[39] **Sheng, D.**, **Yuan, J.**, **Xie, Q. and Luo, P.** (2020). MOOCRec: An Attention Meta-Path Based Model for Top-K Recommendation in MOOC, *Knowledge Science, Engineering and Management: 13th International Conference, KSEM 2020, Hangzhou, China, August 28–30, 2020, Proceedings, Part I*, Springer-Verlag, Berlin, Heidelberg, p.280–288, https://doi.org/10.1007/978-3-030-55130-8_25.

[40] **Xu, S.**, **Yang, C.**, **Shi, C.**, **Fang, Y.**, **Guo, Y.**, **Yang, T.**, **Zhang, L. and Hu, M.** (2021). Topic-Aware Heterogeneous Graph Neural Network for Link Prediction, *Proceedings of the 30th ACM International Conference on Information amp; Knowledge Management*, CIKM '21, Association for Computing Machinery, New York, NY, USA, p.2261–2270, https://doi.org/10.1145/3459637.3482485.

[41] **Han, Z.**, **Anwaar, M.U.**, **Arumugaswamy, S.**, **Weber, T.**, **Qiu, T.**, **Shen, H.**, **Liu, Y. and Kleinsteuber, M.** (2020). Metapath- and Entity-aware Graph Neural Network for Recommendation, *CoRR*, *abs/2010.11793*, https://arxiv.org/abs/2010.11793, 2010.11793.

[42] **Agrawal, R.**, **Srikant, R.** *et al.* (1994). Fast algorithms for mining association rules, *Proc. 20th int. conf. very large data bases, VLDB*, volume1215, Santiago, Chile, pp.487–499.

[43] **Likas, A.**, **Vlassis, N. and Verbeek, J.J.** (2003). The global k-means clustering algorithm, *Pattern recognition*, *36*(2), 451–461.

[44] **Grover, A. and Leskovec, J.** (2016). node2vec: Scalable feature learning for networks, *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp.855–864.

[45] **Mikolov, T.**, **Sutskever, I.**, **Chen, K.**, **Corrado, G. and Dean, J.** (2013). Distributed Representations of Words and Phrases and their Compositionality, *CoRR*, *abs/1310.4546*, `http://arxiv.org/abs/1310.4546`, `1310.4546`.

[46] **Chen, T. and Guestrin, C.** (2016). XGBoost: A Scalable Tree Boosting System, *CoRR*, *abs/1603.02754*, `http://arxiv.org/abs/1603.02754`, `1603.02754`.

[47] **Data61, C.** (2018). *StellarGraph Machine Learning Library*, `https://github.com/stellargraph/stellargraph`.

[48] **Hamilton, W.L.**, **Ying, R. and Leskovec, J.** (2017). Inductive Representation Learning on Large Graphs, *CoRR*, *abs/1706.02216*, `http://arxiv.org/abs/1706.02216`, `1706.02216`.

[49] **Devlin, J.**, **Chang, M.W.**, **Lee, K. and Toutanova, K.** (2018). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, `https://arxiv.org/abs/1810.04805`.

[50] **Grootendorst, M.** (2022). *BERTopic: Neural topic modeling with a class-based TF-IDF procedure*, `https://arxiv.org/abs/2203.05794`.

[51] **Islek, I. and Oguducu, S.G.** (2020). A Hybrid Recommendation System Based on Bidirectional Encoder Representations, ***I. Koprinska, M. Kamp, A. Appice, C. Loglisci, L. Antonie, A. Zimmermann, R. Guidotti, Ö. Özgöbek, R.P. Ribeiro, R. Gavaldà, J. Gama, L. Adilova, Y. Krishnamurthy, P.M. Ferreira, D. Malerba, I. Medeiros, M. Ceci, G. Manco, E. Masciari, Z.W. Ras, P. Christen, E. Ntoutsi, E. Schubert, A. Zimek, A. Monreale, P. Biecek, S. Rinzivillo, B. Kille, A. Lommatzsch and J.A. Gulla***, *editors, ECML PKDD 2020 Workshops*, Springer International Publishing, Cham, pp.225–236.

[52] **Chollet, F.** (2015). *keras*, `https://github.com/fchollet/keras`.

[53] *HuggingFace*, `https://huggingface.co/docs/transformers/model_doc/bert`.

[54] **Wu, Y.**, **DuBois, C.**, **Zheng, A.X. and Ester, M.** (2016). Collaborative Denoising Auto-Encoders for Top-N Recommender Systems, *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining*, WSDM '16, Association for Computing Machinery, New York, NY, USA, p.153–162, https://doi.org/10.1145/2835776.2835837.

CURRICULUM VITAE

Name SURNAME:
Yaren YILMAZ

EDUCATION:

- **B.Sc.:** 2020, Dokuz Eylul University, Faculty of Engineering, Department of Computer Engineering
- **M.Sc.:** 2022, Istanbul Technical University, Faculty of Computer and Informatics Engineering, Department of Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2021-... Research and Teaching Assistant at ITU Artificial Intelligence and Data Engineering Department

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Yılmaz Y.**, ÖĞüdücü, Ş. G. (2021, November). Enriching demand prediction with product relationship information using graph neural networks. *In Proceedings of the 2021 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining* (pp. 561-568). (Accepted)
- Kaya, K., **Yılmaz, Y**, Y., Yaslan, Y., Öğüdücü, Ş. G., Çıngı, F. (2022). Demand forecasting model using hotel clustering findings for hospitality industry.*Information Processing Management,* 59(1), 102816. (Accepted)