Proactive Return Management in E-commerce Supply Chains: Predictive Analytics Approach

by

Tuğçe Uzer

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Industrial Engineering and Operational Management

KOÇ
ÜNİVERSİTESİ

August 17, 2023

Proactive Return Management in E-commerce Supply Chains: Predictive Analytics Approach

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Tuğçe UZER

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Prof. Metin Türkay (Advisor)

_____

Assist. Prof. Kübra Tanınmış Ersüs

_____

Assist. Prof. Utku Koç

Date: _____

To my family


…

ABSTRACT

Proactive Return Management in E-commerce Supply Chains: Predictive Approach

Tuğçe UZER

Master of Science in Industrial Engineering

August 17, 2023

In e-commerce, product returns management has become a critical concern for online retailers. With the drastic growth of online shopping, the increasing volume of returns poses significant challenges to the efficiency of supply chain operations. To effectively address these challenges, proactive return management strategies are essential. This MS thesis aims to predict product returns before the product is sold, leveraging open-source customer-item rating and feedback data. The study adopts a novel approach combining Natural Language Processing (NLP) techniques, Bayesian personalized ranking, ensemble learning, and deep learning methodologies to uncover latent factors behind shopping and rating behavior to make accurate return predictions, aiding online retailers in proactive decision-making and optimizing their supply chain and inventory management processes.

ÖZETÇE

E- Ticaret Tedarik Zincirlerinde Proaktif İade Yönetimi: Öngörücü Yaklaşım

Tuğçe UZER

Endüstri Mühendisliği, Yüksek Lisans

17 Ağustos 2023

E-ticarette, ürün iade yönetimi, çevrimiçi perakendeciler için kritik bir endişe kaynağı olmaya başlamıştır. Çevrimiçi alışverişin hızlı artmasıyla birlikte artan ürün iade hacmi, tedarik zinciri operasyonlarının verimliliği açısından önemli zorluklar doğurmaktadır. Bu zorlukları etkin bir şekilde yönetebilmek için proaktif iade yönetimi stratejileri esastır. Bu yüksek lisans tezi, açık kaynaklı müşteri-ürün değerlendirmesi ve geri bildirim verilerini kullanarak ürün satılmadan önce ürün iadelerini tahmin etmeyi amaçlamaktadır. Çalışma, muhtemel ürün iade tahminlerini yapmak amacıyla müşteri alışveriş davranışının arkasındaki gizli faktörleri ortaya çıkararak çevrimiçi perakendecilere proaktif karar vermede yardımcı olmayı hedeflerken Doğal Dil işleme (NLP) tekniklerini, Bayes kişiselleştirilmiş sıralamasını (BPR), kolektif öğrenme ve derin öğrenme metodolojilerini birleştiren yeni bir yaklaşım benimsemektedir.

AKNOWLEDGEMENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

NLP	Natural Language Processing
BPR	Bayesian Personalized Ranking
MF	Matrix Factorization
DNN	Deep Neural Network
RL	Reverse Logistics
ANN	Artificial Neural Network
SMOTE	Synthetic Minority Over-sampling
RUS	Random Under Sampling
RF	Random Forest
XGB	XGBoost

Chapter 1:
INTRODUCTION

A supply chain is an organized network of suppliers, production facilities, retailers, and transportation routes to acquire raw materials efficiently, transform them into finished goods, and distribute them to customers (Pishvaee et al., 2011). Due to running out of resources and environmental problems, this definition must be extended by including reverse flows (Paksoy & Özceylan, 2012).

Besides resource limitations and regulations, there are different technological evolutionary reasons, such as increased e-commerce users, number of returned items, and consumer awareness. The help of these reasons has led firms in all sectors that manage distribution processes to revise their logistics management.

Traditionally, return management in the supply chain has been retroactive, focusing on handling returns after they occur. However, the shifting dynamics of the e-commerce landscape demanded a proactive approach to return management. This proactive approach entails identifying potential return instances before the purchase. By predicting product returns in advance, online retailers can implement preemptive measures, optimize inventory management, and streamline their supply chain processes.

Product returns disrupt the smooth flow of goods within the supply chain and impose substantial costs on retailers. The reverse logistics associated with returns involve intricate processes such as handling, inspection, refurbishment, and restocking. These activities consume resources, add logistical complexities, and may lead to inventory obsolescence. Additionally, returns impact customer satisfaction, potentially resulting in loss of future sales and damage to brand reputation. Consequently, implementing efficient return management strategies minimizes operational costs and maximizes customer loyalty.

Studies presented that optimal return management leads to cutting costs, increasing customer satisfaction, and a higher competitive advantage (Dutta et al., 2020; Ramanathan, 2011; Röllecke et al., 2018). Forecasting demand-return patterns have significant value in supply chain operations because these forecasts are the strategic center of all operational decisions, such as operations in distribution centers, transportation planning, and inventory handling plans. So, having a well-functioning logistics network requires correctly forecasted demand-return patterns to increase firms' market share and to obtain a more sustainable environment.

Predictive modeling, supported by advanced data analytics techniques, enables proactive return management. Leveraging historical customer data, including customer-item ratings – feedback, and any historical return data, can be extracted to understand customer sentiments and predict return probabilities. By accurately predicting return probabilities, retailers can optimize inventory levels, identify potential quality issues, and implement targeted marketing strategies to reduce returns. Additionally, anticipating return instances allows for improved planning of reverse logistics operations, resulting in more efficient resource allocation and reduced costs.

To provide further clarification, the rest of the thesis is structured as follows; related studies in the literature review in Chapter 2, problem and solution approach in Chapter 3, outcomes of the model in Chapter 4, and actionable insights and future research areas in Chapter 5 as the conclusion.

Chapter 2:

LITERATURE REVIEW

Return management is defined as one of the essential supply chain procedures for management and operations associated with returns, gatekeeping, and logistics control within the company and among supply chain members (Rogers et al., 2002). Managing returns efficiently in a forward-reverse integrated supply chain obtains a significant competitive advantage for a firm over other firms. To gain a competitive advantage, firms must determine the industry's attributes and find feasible, even optimal, strategies to preserve a well-functioning supply chain with whole aspects such as inventory management, return management, information management, etc.

So, all management processes must be designed considering industry-specific characteristics. Like all other industries, e-commerce has different dynamics than selling products via brick-and-mortar channels or Omni-channels. Even though the efficiency of forward-reverse supply chains seems similar or comparable for e-commerce and physical retailers, reverse logistics is much more critical in e-commerce retailers. Traditional retailing considers the product availability on the shelf to prevent lost sales, while e-commerce retailing considers non-return sales because there is no try-and-buy option. This unique aspect of e-commerce modifies the critical balance between forward and reverse logistics.

Furthermore, e-commerce has accelerated growth, considering the ease of reaching substitute products, high competition between retailers and marketplaces, and increased internet usage, especially during the global pandemic. The number of users and newly entered retailers in marketplaces have outstandingly increased in a couple of years. Statista investigated the volume of worldwide retail e-commerce sales between 2014 and 2020, showing a 228% growth (Global retail e-commerce market …", 2021).

While distinctive characteristics of e-commerce retailing add complexities to its supply chains, the high penetration rate in the spread of the Internet and e-commerce retailing also brings a high return rate of 30% (Dennis, 2018). These high return rates add more complexity to e-commerce supply chains than wholesale or retail

merchandising. These complexities make the e-commerce industry more different, with different challenges. Considering these dynamics, return management has become a crucial part of reverse logistics in the supply chain for an e-commerce retailer (Nanayakkara et al., 2022).

In e-commerce, return management is crucial in ensuring customer satisfaction and maintaining a successful business. In a reverse logistics context, there are many ways to manage returns practically. Two primary approaches to sustain return management are proactive and retroactive approaches. Both proactive and retroactive approaches are essential for sustaining return management in e-commerce. The proactive approach focuses on prevention and reducing the occurrence of returns, while the retroactive approach focuses on managing returns efficiently and using return data to drive improvements. By combining these approaches, e-commerce retailers can create a comprehensive return management strategy that promotes customer satisfaction, reduces costs, and enhances overall business performance.

The retroactive approach deals with managing returns that have already occurred. It focuses on streamlining the return process, addressing customer concerns, optimizing reverse logistics network design, and finding ways to minimize the impact of returns on the business. This approach involves efficient return handling procedures, such as providing clear return instructions, offering flexible return policies, and implementing hassle-free return processes. Additionally, the retroactive approach involves analyzing return data to identify patterns and trends, understanding the reasons behind returns, and using this information to improve product design, inventory management, and customer service. By effectively managing returns, the retroactive approach aims to eliminate the negative impact of returns on customer satisfaction and overall business performance. Many studies elaborated on this approach with different optimization methods and reported outperforming findings to optimize reverse logistics costs.

- Regardless of the company size, companies decide to invest considerable budgets in setting up their transportation processes as an R&D project or sign contracts with third-party logistics firms to eliminate the complexity of return management. Besides giving responsibility to third-party logistic firms or allocating enormous budgets for reverse networks, businesses have formed

different solutions for their reverse logistics to keep returns under control. Canceling the free-return policy is one option that enterprises can follow. However, e-commerce customers' shopping decisions are affected mainly by two crucial features; free shipping and a free return policy in the marketplace when they decide to shop (Saleh., 2022).

- Building drop-out points in commonly crowded areas as pick-up points for the firm is a solution for cost-cutting; however, the solution can only apply to some customers' behavior patterns and locations. Chang & Zheng's (2014) suggestion to cut costs in non-defective reverse logistics is outstanding. They claimed that clients might reduce the distance of return transportation by using the delivery address of another customer as their return address. According to simulation and computation tests for their study, the proposed technique can reduce the cost of the consumer's return transportation (Chang & Zheng, 2014).

- Sending back the bulk of returns to the initial depot instead of returning the product as it arrives is another option to minimize reverse logistics costs. Das et al. (2020) conducted a study to design the reverse logistics network specifically for an e-commerce company specializing in fashion goods. The study focused on achieving a balance between cost and responsiveness. The researchers identified several potential nodes called Initial Collection Centers to achieve this. These centers were responsible for initially collecting and storing customer returns for a defined period before they were transported to the final warehouses. The main aim was to minimize the overall cost associated with reverse logistics operations, encompassing various stages such as collection, inspection, storage, and disposition of returns (Das et al., 2020). The study also involved;
    - Determining the optimal locations for the Initial Collection Centers
    - Assigning customer returns to the appropriate centers
    - Determining the volume of returns to be transported in each shipment to the final warehouse.

- In e-commerce, returned products may cause a misusage or already damaged items delivered to the user. Firms must understand and segment the reasons for

returns to save time and operational efficiency by constructing collection, inspection, storing, and disposal centers. Li et al. (2014) developed a two-stage algorithm to address the closed-loop location inventory problem in e-commerce, considering returns. Their approach involved combining recycling and distribution centers to create Merchandise Centers (MCs) that served as dual-purpose facilities with an additional inspection function. The main objective of the model was to minimize the overall cost associated with both forward and reverse logistics networks while simultaneously determining optimal quantities, locations, order times, and order sizes for the MCs. To solve this complex nonlinear mixed programming model, the researchers proposed an effective two-stage heuristic algorithm, Lagrangian relaxation, combined with an ant colony algorithm (LRCAC), and the algorithm provided a practical and efficient solution to the problem (Li et al., 2014).

Even though there are increasing e-commerce retailers, users, and marketplaces, there are still significant gaps in the literature on constructing optimal reverse network design (Das et al., 2020). Even though many studies have elaborated on the topic and offered many network designs and models, the topic is still open to research. One of the reasons behind that gap is that most studies from a retroactive perspective elaborated on the topic to optimize return collection and minimize reverse logistics costs. Besides these strategies, collected shopping data utilizing business-specific know-how can be analyzed to understand users' shopping and returning behavior. The analysis and insights can help to predict users returning patterns and minimize reverse logistics costs proactively.

The proactive approach focuses on preventing returns from happening in the first place. It involves taking preventive measures and implementing strategies to minimize the likelihood of returns. This approach includes various strategies such as improving product descriptions and images to provide accurate information to customers, enhancing sizing guides and fit recommendations, optimizing product quality and packaging to minimize defects, and offering customized recommendations based on user preferences and past purchases. Addressing potential issues before the purchase is

made, the proactive approach aims to reduce the need for returns, leading to improved customer satisfaction and cost savings for the e-commerce retailer.

A proactive strategy in return management using data is a newly studied topic with different methods and aims. While some studies used time-series models to forecast return volume (Toktay, 2001), some researchers elaborated more sophisticated methods. Urbanke et al. (2015), Mahalanobis Feature Extraction was employed to forecast product returns in the e-commerce domain. However, it should be noted that predictions could only be generated after the user had already placed an order, limiting the e-tailer's ability to implement proactive measures to mitigate returns. Furthermore, this approach was designed differently for user-product level prediction, which may limit its effectiveness in providing detailed insights at that level of granularity.

E-commerce retailers often need help to decrease high return rates, resulting in increased costs and diminished profit margins. Kedia et al. (2019) presented a novel approach to detect returns and implement specific actions to address this issue. The proposed method utilizes a hybrid dual model, constructed with a deep neural network, to detect returnable carts and products in real-time based on the user's current cart configuration. The innovative methodology leverages Deep Neural networks (DNN). To capture users' preferences and the underlying latent features of products, we employ product embeddings derived from Bayesian Personalized Ranking (BPR). Additionally, the study utilizes a separate set of embeddings to capture users' body shape and size, employing a skip-gram-based model. These embeddings, alongside engineered features, are integrated into the DNN to forecast the probability of returns. The experimental results regarding the action items demonstrate that accurate return prediction can effectively reduce the return rate (Kedia et al., 2019).

Besides improving guidelines about product textiles, size, and fit or visuals with better resolutions, accurately predicting customers' size and fit from their feedback is also another sophisticated way to minimize returns because of that reason. However, accurately predicting product size recommendations and fit feedback from customers presents significant challenges due to the feedback's nuanced nature and the labels' imbalanced distribution. To address these issues, Misra et al. (2018) introduced a novel

predictive framework that effectively captures the underlying semantics of customers' fit feedback. In addition, a metric learning technique addresses label imbalance problems. Qualitative and quantitative study results showed that predicting customers' size and fit semantics affects return rates significantly (Misra et al., 2018). Furthermore, the researchers contribute two publicly available datasets from online clothing retailers, which can serve as valuable resources for further research in this domain. This thesis study uses one of the open-sourced datasets as the primary data.

Chapter 3:
PROBLEM DESCRIPTION AND MODELING

3.1 Problem Description

These days, proactive return management has transformed from a nice-to-have status into a need status. Traditionally, return management has been retroactive, focusing on handling returns after they occur. However, the dynamics of the e-commerce landscape, characterized by increased e-commerce users, the volume of returned items, and consumer awareness, necessitate a shift towards a proactive approach.

However, managing return operations proactively is a challenging task to obtain a sustainable success rate in the long term. Predicting return behavior in e-commerce poses significant challenges due to various factors. Direct feedback in online transactions is necessary to assess customer preferences accurately. Moreover, the complex and diverse nature of customer behavior, influenced by factors like product features, price, and personal preferences, adds complexity to the prediction task. Data sparsity and class imbalance issues in e-commerce datasets hinder the development of accurate predictive models.

Additionally, the abundance of unstructured data, such as customer feedback and reviews, requires advanced NLP techniques for meaningful analysis. The dynamic nature of e-commerce, with evolving trends and preferences, necessitates adaptive models to capture the latest patterns accurately. Lastly, external factors like seasonality and competitor actions impact return rates, further complicating the prediction process. Addressing these challenges requires sophisticated analytics techniques, robust data preprocessing, and a deep understanding of e-commerce customer behavior.

3.2 Data Description

This study and all other return management operations must have a well-compiled product return dataset to predict return probabilities accurately. Acquiring return data in e-commerce can be challenging for several reasons. Return data may be stored separately or not maintained with sufficient granularity, limiting availability and efficiency. Even if stored in a proper format, retailers and e-commerce platforms may need more time to readily provide access to detailed return data due to privacy concerns or proprietary reasons. Additionally, the confidential nature of return data, containing sensitive customer information, also restricts its accessibility.

Since there is no return dataset, this study has focused on other relevant data sources, such as customer feedback, ratings, or product reviews, to gain insights into return behavior and improve return management strategies.

We used an open-source dataset that contains e-commerce clothing purchases. The dataset was collected from online clothing retailers for a specific study to improve customer shopping experiences and reduce product return rates through accurate product size recommendations and fit prediction (Misra et al., 2018). The dataset has 82790 rows and 18 columns, presenting rating, feedback, and item-related features. The dataset contains 47958 unique users, 1378 unique items, and ratings ranging from 0 to 5, named as quality in the data frame. Table 3.2.1 shows the columns, data types, and the number of non-null values in the corresponding columns.

Data Shape : (82790,18)
Data columns (total 18 columns):

#	Column	Non-Null Count	Dtype
0	item_id	82790 non-null	object
1	waist	2882 non-null	object
2	size	82790 non-null	int64
3	quality	82722 non-null	float64
4	cup size	76535 non-null	object
5	hips	56064 non-null	object
6	bra size	76772 non-null	object
7	category	82790 non-null	object
8	bust	11854 non-null	object
9	height	81683 non-null	object
10	user_name	82790 non-null	object
11	length	82755 non-null	object
12	fit	82790 non-null	object
13	user_id	82790 non-null	object
14	shoe size	27915 non-null	object
15	shoe width	18607 non-null	object
16	review_summary	76065 non-null	object
17	review_text	76065 non-null	object

Table 3.2.1: Dataset Information

Even though Metadata contains ratings, reviews, user-item measurements, and category info, the study will cover only ratings, reviews, fit info, category info, and user-item ids. Ratings have been used to reach user-item embeddings through Bayesian personalized ranking - matrix factorization. For this analysis, we had to change the rating values of the evaluated rows from zero to one and assumed that the ratings were between one and five. This assumption is essential for Bayesian Personalized Ranking, which will be discussed later. Figure 3.2.1 shows the possible rating values and corresponding counts in the dataset.

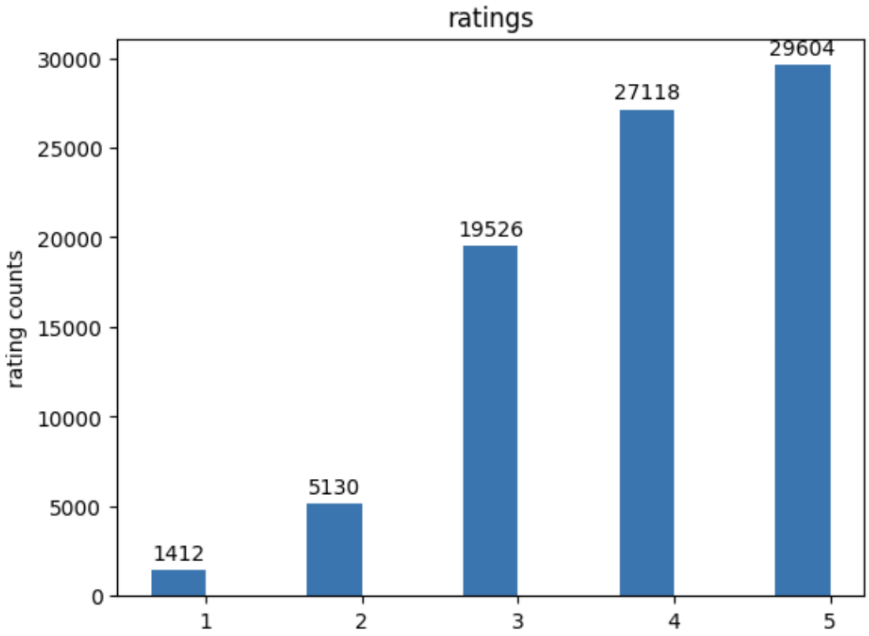


Figure 3.2.1:Counts of the ratings in the dataset

Reviews have been used to create a new column that shows product returns or not returns through analyzing sentiments in customer reviews. In addition, user-item measurements and category info have been used to have engineered features for final raw data.

3.3 Solution Approach

Since the dataset has no return column, we created a return column by deriving review columns as a preprocessing step. Firstly, return labeling is performed using NLP sentiment analysis on customer reviews. This process involves analyzing the text to determine the sentiment expressed by the customers towards the product. The model gains insight into customer satisfaction by labeling the target variable based on sentiment (e.g., positive or negative) using reviews. The obtained return column showed that the dataset is imbalanced with 14% return labels. Detailed explanation about the target labelling is explained in chapter 3.3.1. Regarding labelling results, we apply oversampling and under-sampling methods to overcome the imbalanced dataset drawbacks in ML models to reach a resampled balanced dataset besides the original dataset.

Then, feature extraction incorporates rating, category, and fit information. These features provide additional context and contribute to the model's predictive power. For instance, the rating given by the customer, the product category, and the fit information (e.g., whether the product fits well) can all impact the likelihood of a return. With the help of these steps, we obtained a preprocessed final dataset for different predictive models.

The study used different predictive models to compare the performance: ensemble learning algorithms and deep neural network structures. The study used Random Forest and XGBoost algorithms as ensemble learning techniques and multi-ınput fully connected deep neural networks as a more complex structure.

Decision tree structures are chosen as the base for the first part of the solution approach of the study. However, while simple and interpretable, decision trees can be prone to overfitting, meaning they may perform well on training data but poorly generalize the unseen data. When it comes to decision tree structures, ensemble learning is valuable. Ensemble learning is a well-known and well-functioning machine learning technique. It combines the predictions of multiple individual models to make more accurate and robust predictions. Ensemble methods address this issue by constructing a collection of decision trees and aggregating their outputs. By combining multiple

decision trees, each trained on different subsets of the data or with different parameters, ensemble methods reduce the risk of overfitting and improve the model's ability to capture complex relationships in the data. Ensemble learning is a critical tool in machine learning because it improves predictive accuracy and boosts model reliability and stability, making it indispensable for many real-world applications where decision trees alone may fall short.

In ensemble learning, bagging and boosting are two fundamental techniques used to improve the performance of machine learning models, especially decision trees. Both methods aim to improve model performance by leveraging multiple base learners but doing so in different ways. While bagging reduces variance and improves stability, boosting reduces bias and focuses on correcting misclassifications.

Bagging (Bootstrap Aggregating):

Bagging is a technique that aims to decrease the model's variance by training multiple instances of the same base learner on different subsets of the training data and then averaging their predictions (for regression) or using majority voting (for classification). The technique can be explained as in the following steps:

- Bootstrap Sampling: Bagging starts by creating multiple random subsets of the training data through a process called bootstrap sampling. Each subset is created by randomly selecting data points with replacements from the original training dataset. This means some data points may appear in multiple subsets while others may not.
- Parallel Model Training: A base learner (decision tree) is trained independently on these subsets. Since each subset is slightly different, the models will have some variation in their predictions.
- Aggregation: For regression tasks, the final prediction is obtained by averaging the predictions of all base models. Classification tasks involve majority voting, where the most common class prediction among all base models is chosen as the final prediction.

This study uses Random Forests as the bagging algorithm, an ensemble of decision trees trained with bagging. Bagging helps reduce overfitting and improves the stability and accuracy of the model. The Random Forests takes the extracted features and return

labels from the preprocessing steps and combines them meaningfully. Then, the model is tuned with Grid Search to obtain the best hyperparameters.

Boosting:

Boosting is another ensemble technique that focuses on improving a model's bias by iteratively training a sequence of base learners, with each learner giving more attention to the data points that the previous ones misclassified. The technique can be explained in the following steps:

- Sequential Model Training: Boosting starts with a base learner (a weak decision tree) and trains it on the entire training dataset. After training, the model's performance is evaluated.
- Weighted Data: Data points that the current model misclassified are given higher weights, effectively making them more important in the next round of training. This allows the subsequent base learner to focus on the previously misclassified samples.
- Iterative Process: The process is repeated for a specified number of rounds or until a predefined criterion is met. Each new base learner is trained with the updated data weights.
- Final Prediction: The final prediction is usually obtained by combining the all base models' predictions, with each model's contribution weighted based on its performance during training.

This study uses XGBoost as the boosting algorithm, an ensemble of decision trees trained with boosting. Boosting tends to reduce bias and variance, often leading to very accurate models, but it can be more prone to overfitting if not correctly tuned. In the study, the XGBoost Regressor takes the extracted features and return labels from the preprocessing steps and combines them meaningfully. The model is tuned with Grid Search to find optimal hyperparameters.

Besides ensemble learning, we built a deep neural network structure to capture better performance. In the DNN structure, we build a stepwise model to predict whether the product will be returned. As an additional step to feature engineering in base preprocessing steps, we applied a different method to capture user-item shopping

behavior. User item embeddings are obtained to uncover the latent factors driving rating behavior. Embeddings are achieved through Matrix Factorization based on Bayesian Personalized Ranking (BPR) loss, a collaborative filtering technique. By considering user preferences and item characteristics, BPR captures the underlying patterns influencing customers' ratings, helping to understand their decision-making process.

Then, a multi-input, fully connected deep neural network with two hidden layers is constructed to predict whether a product will be returned. This neural network takes in the extracted features, embeddings, and return labels from the previous steps and combines them meaningfully. By training on a labeled dataset, the model learns to predict based on the input features, ultimately estimating the likelihood of Return for a given product.

In conclusion, the study elaborated the following predictive models and resampling techniques with corresponding sample sizes.

	ORIGINAL DATASET	UNDERSAMPLING (RANDOM UNDER SAMPLING)	OVERSAMPLING (SMOTE)
RANDOM FOREST	82790 Samples	35346 Samples	130234 Samples
XGBOOST	82790 Samples	35346 Samples	130234 Samples
DNN	82790 Samples	35346 Samples	130234 Samples

Table 3.3.1: Summary table of predictive models in the study

3.4 Preprocessing Steps

3.4.1 Preprocessing Steps – NLP Based Target Labelling

Natural Language Processing (NLP) is a research area that centers around the interaction between computers and human language. Its goal is to create algorithms and models that empower computers to comprehend, interpret, and generate human language in a meaningful manner. NLP encompasses diverse tasks, including text classification, language translation, information extraction, and sentiment analysis. By harnessing the power of NLP, computers can process and derive valuable insights from textual data, leading to advancements in various fields and applications.

Sentiment analysis is a specific application of NLP to determine a text's sentiment or emotional tone. It involves analyzing the subjective information conveyed by words and phrases to classify the overall sentiment as positive, negative, or neutral. Sentiment analysis can be performed at different layers, from individual words or sentences to entire documents or conversations. For this study, we analyzed many sentences written as reviews for items.

Since the dataset has two columns filled with reviews, the NLP package runs for both separately. As a side note, both columns have NaN values as blank at the same indices, which compose almost nine percent of the data. The first column shows review summaries mainly composed of one short sentence, and the sentiment behind it is hard to catch. For example, one review summary stated, *'It is a beautiful jacket!'*. However, the second column has more detailed reviews, named review_text in the dataset, such as *'It is a beautiful jacket! I love how it is knit...'*. Since the review text column gives a more detailed explanation, analyzing that column helped to catch more accurate sentiment behind the reviews.

Sample sentiment score results are listed in the following. First review is taken from the review_summary column and second review is taken from the review_text column and corresponding sentiment scores are taken from algorithm results.

Review: I really wanted to love t.
Sentiment Scores: {'neg': 0.0, 'neu': 0.402, 'pos': 0.598, 'compound': 0.6666

Review: I just have to say that I LOVE this blazer! It's perfect for the fall...warm and the color is prefect. The medium fits like a glove on me.
Sentiment Scores: {'neg': 0.0, 'neu': 0.658, 'pos': 0.342, 'compound': 0.9086}

To analyze the review summary column, the Python NLTK package was employed. The package was executed to analyze the sentiment of each review and generate a compound score. The compound scores consider the following calculated scores, negativity, neutrality, and positivity. Based on the compound score, we labeled each purchase as a return or not, considering the following conditions:

- If the compound score is less than zero, the negativity in the review is much higher, and the corresponding item is more likely to be returned. That is why the corresponding index in the return column changed to 1, indicating it is a return. The preprocessing review summary column showed that 3620 purchases have negative compound scores. Furthermore, the corresponding 3620 indices are labeled as returns in the target column.

- If the compound score is greater than zero, the positivity in the review is much higher, and the corresponding item is unlikely to be returned. That is why the corresponding return value is assigned as 0, indicating it is not a return.

- If the compound score is zero, sentiment analysis shows pure neutrality. The neutral sentiment means null values in our case. After analyzing the review text column, a robust approach is executed for those purchases. In total, 6725 purchases have no review or feedback in the dataset.

The same preprocessing approach was executed for the review text column also. Considering the same logic, we labeled 3699 indices as returns also. For purchases without review, the rating could be an essential key to mark a return or not return. If the rating is less than 5, the purchase return status is labeled as return, otherwise labeled as not return. Employing this robust approach, we assigned one as a return, 4423 out of 6725 purchases.

To conclude, each review and purchase have a return status, labeled by sentiment-based or robust approaches. In total, 11742 purchases were considered as returns out of 82790 purchases. The following figure shows the dispersion of return labels by method and the corresponding percentage of return status and not return status in the dataset. We re-designed the dataset with preprocessing and held a purchase return dataset with an almost 14 percent return rate.

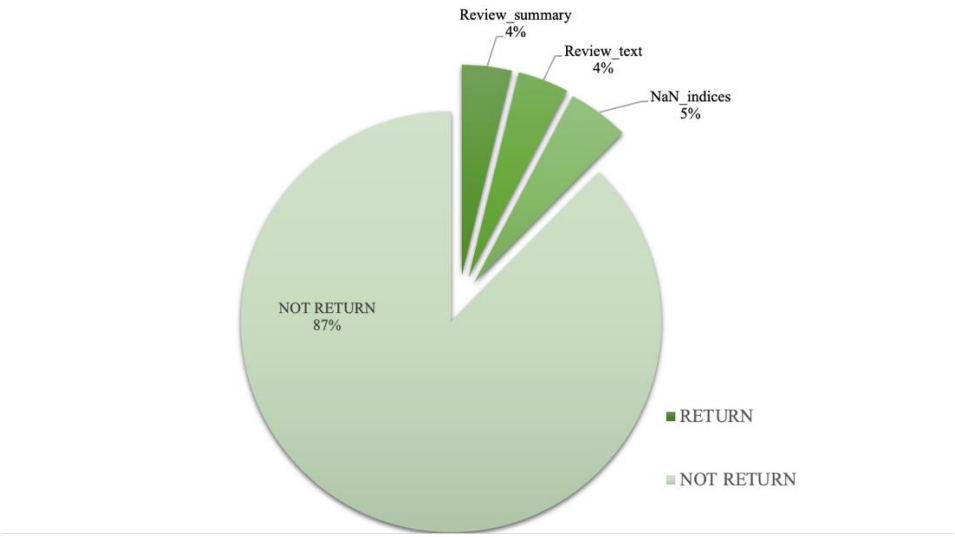


Figure 3.4.1:Dispersion of return status

3.4.2 Preprocessing Steps – Feature Engineering

The importance of this feature extraction step lies in its ability to provide the predictive model with relevant and informative signals about customer behavior and item characteristics. By leveraging these extracted features, the network can effectively learn the complex relationships between the features and the target variable, which is the prediction of the product return. This leads to improved prediction accuracy and the ability to identify the underlying factors driving return behavior in e-commerce. Additionally, incorporating item category features enables the network to consider the contextual information associated with different product categories. By capturing the characteristics and attributes specific to each category, the network gains a more comprehensive understanding of the rated items and their potential impact on return behavior. This allows the model to account for any patterns in customer preferences and return tendencies across different product categories.

The feature extraction step from rating data and item category information captures essential aspects of customer behavior, item characteristics, and contextual information. It enables the deep neural network to leverage these features and make more informed and accurate predictions regarding product return in e-commerce, ultimately contributing to enhanced decision-making and improved supply chain management.

Reusing rating data as input in both BPR and the neural network can lead to data leakage and then overfitting. Data leakage refers to a situation where information from the target variable inadvertently leaks into the training data, allowing the model to make predictions based on this leaked information. This can result in overly optimistic performance metrics during training but may lead to poor generalization on new, unseen data.

In this study, if the rating data is directly used as input in the neural network alongside the return labels, there is a risk that the network may learn to exploit direct correlations between the ratings and the return labels. This would introduce data leakage, as the model would have access to information about the return behavior that it should not have during prediction. This is prevented by excluding the direct rating values from the input features and focusing on other relevant information that is not

directly related to return behavior. This is where feature engineering plays a crucial role, as it allows for creating meaningful features that capture user-item interactions, item characteristics, user preferences, and contextual information without directly leaking information about return behavior.

Considering the above reasons, we preferred to add the feature engineering step as a preprocessing step for the deep neural network model.

The dataset used in this step includes several columns that capture important information about the users, items, ratings, categories, fit, and return status. The "user_id" column contains unique identifiers assigned to each user, ranging from 1 to 47,958. Similarly, the "item_id" column contains unique identifiers assigned to each item, ranging from 1 to 1,378. The "rating" column represents the rating users give to the items on a scale of 1 to 5.

The "category" column provides information about the category to which each item belongs, such as dresses, weddings, sales, tops, bottoms, and outerwear. This allows for understanding the specific type or style of the items. The "fit" column indicates the fit of the items as reported by the users, with values including small, large, or possibly other fit descriptions.

Lastly, the "return" column indicates whether an item was returned or not. It is represented by binary values, where 0 signifies that the item was not returned, and one indicates that the item was returned. This column plays a crucial role in the prediction task, as the goal is to predict the return status of items based on the given information. The following figure shows a brief outlook for the columns in the raw dataset.

	user_id	item_id	rating	category	fit	return
values	[1:47958]	[1:1378]	[1:5]	dresses	small	0: Keep
				wedding	fit	1: Return
				sale	large	
				tops		
				bottoms		
				outerwear		

Table 3.4.1:Raw dataset for the feature extraction step

One of the engineered features in this study is an item's average rating deviation with respect to its category's mean rating. This feature provides insights into how the rating of an item compares to the average rating of other items in the same category. The calculation of this feature involves subtracting the overall average rating of item i from the average rating of the corresponding category mean of item i. The feature is denoted as 'avg_item_behav_cat' in the final dataset.

$$value\_item_{ij} = Mean\ rating\ of\ category\ j - Mean\ rating\ of\ item\ i$$

If the value of this feature is greater than zero, it indicates that the item has a lower rating than the average rating of other items in its category. This could imply several possible insights. For instance, it may suggest that the item i has some defects or quality issues, leading to lower ratings for other items in the same category. Another possibility is that the item may be misrepresented in size or fit, causing customers to rate it lower.

Conversely, suppose the value of this feature is greater than zero. In that case, it suggests that item i has a higher rating than the average rating of other items in its category. This could indicate that the item performs exceptionally well within its category, potentially offering superior features, quality, or customer satisfaction.

Incorporating this engineered feature into the predictive model allows for capturing an item's relative rating position within its category. This information can help to understand potential issues or strengths associated with items, aiding in decision-making processes related to inventory management, product improvements, or marketing strategies.

Another engineered feature in this study focuses on understanding customer behavior when rating an item. This feature calculates the difference between the rating given by a user for item i and the average rating of item i in the dataset. This feature aims to capture how a particular user's rating behavior deviates from the average behavior of other users for the same item. The engineered feature is denoted as 'cust_behav_item' in the final dataset.

When the value of this feature is greater than zero, it suggests that the user tends to rate the item higher than other users on average. This could indicate a positive bias or a higher inclination to give favorable ratings. Conversely, when the value is lower than zero, the user is less likely to rate the item higher than other users. This could indicate a more critical or discerning rating behavior.

Understanding user rating behavior is crucial as it reflects their subjective preferences and tendencies. Everyone may have their unique rating habits and criteria when evaluating items. By considering this feature, the study aims to capture the specific rating behavior of users and incorporate it into the predictive model. This information can provide insights into user preferences, satisfaction levels, and potential biases, enabling personalized recommendations and targeted marketing strategies tailored to individual user behavior.

$$value\_user_{ij} = rating\ of\ item\ i\ by\ user\ j - Mean\ rating\ of\ item\ i$$

The final preprocessing step is to transform categoric features. In order to incorporate the categorical features of "category" and "fit" into the predictive model, a transformation process is applied using dummy variables. This process involves converting the categorical values into binary values of 0 and 1, which the model can easily understand and utilize.

For the "category" column, dummy variables are created for each distinct category in the dataset. Each category is assigned a separate binary variable, where a value of 1 shows that the item belongs to that specific category, and a value of 0 indicates it does not. This transformation allows the model to capture the categorical information of the item's category as a numerical representation, enabling it to learn and recognize patterns associated with different categories.

Similarly, the "fit" column is also transformed using dummy variables. The possible fit options (e.g., small, large, fit) are converted into separate binary variables. For each item, the corresponding fit variable is assigned a value of 1 to imply that the

item has that fit characteristic, while the other fit variables are set to 0. This transformation allows the model to consider the fit options as distinct features and account for their impact on the prediction of the return.

The model can effectively utilize this information in its calculations and predictions by transforming the categorical features into binary dummy variables. It enables the model to account for the influence of different categories and fit options on the likelihood of returns, ultimately improving the accuracy and effectiveness of the predictive model in capturing the relationships between these categorical features and the target variable.

Preprocessing steps are cumulated to obtain the final dataset, found below. The below figure shows the final features and structure of the dataset.

#	Column	Count	Null ?	Dtype
1	user_id	82790	non-null	int64
2	item_id	82790	non-null	int64
3	return	82790	non-null	int64
4	avg_item_behav_cat	82790	non-null	float64
5	cust_behav_item	82790	non-null	float64
6	category_bottoms	82790	non-null	int64
7	category_dresses	82790	non-null	int64
8	category_new	82790	non-null	int64
9	category_outerwear	82790	non-null	int64
10	category_tops	82790	non-null	int64
11	category_wedding	82790	non-null	int64
12	fit_fit	82790	non-null	int64
13	fit_large	82790	non-null	int64
14	fit_small	82790	non-null	int64

Table 3.4.2:Final dataset information

3.5 Resampling Data

Resampling is a crucial technique used to address imbalanced datasets in machine learning. An imbalanced dataset occurs when one class (the minority class) is significantly underrepresented compared to the other classes (the majority class or classes). The need for resampling in imbalanced datasets arises due to several important reasons:

- Biased Model Performance: When a dataset is imbalanced, machine learning models tend to be biased towards the majority class. This bias results from the model's inclination to predict the majority class more frequently because it minimizes the overall error rate. As a result, the model may perform poorly in accurately predicting the minority class.
- Loss of Information: When the minority class is severely underrepresented, the model may not have sufficient examples to learn meaningful patterns and relationships within that class. This lack of data can lead to an incomplete understanding of the minority class and suboptimal model performance.
- Misclassification Costs: In some applications, the cost of misclassifying instances from the minority class can be significantly higher than those from the majority class. Resampling helps to mitigate this issue by improving the model's ability to classify minority class instances correctly.
- Model Robustness: Imbalanced datasets can lead to models that are overly sensitive to the data distribution, making them less robust when faced with variations in real-world data distribution. Resampling can help make the model more resilient to changes in class distribution.

To address these issues, resampling techniques are used. There are two main approaches to resampling:

Oversampling: Oversampling involves increasing the number of instances in the minority class. This can be done by duplicating existing minority class instances (random oversampling) or generating synthetic samples (e.g., using SMOTE) to create a more balanced distribution. We preferred to use SMOTE in the study to get more realistic samples derived synthetically from the original samples.

Undersampling: Undersampling is based on reducing the number of instances in the majority class. This can be achieved by randomly removing instances from the majority class, thereby reducing its dominance in the dataset.

Resampling helps balance the class distribution, provides the model with more representative data, and improves the model's ability to make better predictions on minority and majority classes. However, it is essential to carefully evaluate the impact of resampling on model performance and avoid introducing bias or overfitting during the process. That's why, these techniques must be tuned while monitoring proper metrics according to the task.

3.5.1 Oversampling - SMOTE

SMOTE stands for "Synthetic Minority Over-sampling Technique." It is a popular method to address class imbalance in machine learning datasets, mainly when dealing with imbalanced binary classification problems. SMOTE generates synthetic examples for the minority class to balance the class distribution. Following steps explain how SMOTE works.

- Identifying the Minority Class: SMOTE identifies the minority class in the dataset. This is the class that needs to be more represented and needs oversampling. For our case, the underrepresented class is labeled with '1', representing returned items in the dataset.
- Selecting a Minority Class Instance: For each instance in the minority class, SMOTE randomly selects one instance. Let's name this instance as "A."
- Finding Nearest Neighbors: SMOTE calculates the distance between instance A and all other instances in the minority class. The number of nearest neighbors considered ‘k,' is a parameter the user sets.
- Generating Synthetic Samples: SMOTE creates k synthetic samples for each selected instance A by interpolating between instance A and its k nearest neighbors. It does this by selecting a random number ranging from 0 to 1 for each feature and multiplying it by the difference between the feature values of A and its neighbor. These random values are added to A's feature values to create the synthetic samples.

- Example: If A is a point in a two-dimensional feature space, and it has one neighbor, B, SMOTE would create a synthetic sample as follows:
- Synthetic Sample = A + random_value * (B - A)
- This process is repeated for each k nearest neighbor, resulting in k synthetic samples for each selected instance A.
- Balancing the Dataset: After generating synthetic samples for all selected instances in the minority class, the dataset is now more balanced, with an increased number of minority class instances. The original minority class instances and the synthetic samples are combined to create the oversampled dataset.

SMOTE is helpful because it addresses class imbalance by creating synthetic samples representative of the minority class distribution. These synthetic samples help prevent the model from being biased towards the majority class and can improve model performance on imbalanced datasets.

However, it is essential to note that SMOTE has some limitations and considerations:

- SMOTE can introduce noise or overfitting if not used properly, especially when the dataset is imbalanced.
- The choice of the number of nearest neighbors and the method of interpolation can impact the effectiveness of SMOTE, and these parameters need to be tuned carefully.

In summary, SMOTE is a technique for oversampling the minority class by generating synthetic samples based on the existing minority class instances and their nearest neighbors, thereby helping to mitigate class imbalance in classification tasks.

In the study, a train set of 80% of the dataset is oversampled using SMOTE. While implementing SMOTE, reducing the effect of limitations is essential. Tuning the number of neighbors, called k value, is crucial. Choosing small k value as 1,2 leads to overfitting since resampled data becomes full of duplicated original data. On the other hand, choosing big k value such as 9,10 leads to underfitting, since resampling data take

into accounts the further away neighbors. That's why choosing optimal k value is important to get representative resampled data, implementing. We tried different k options to create synthetic samples. Values between 3 and 10 are tried as the number of neighbors (k) in the code, monitoring mean squared error in the regression task. The best number of neighbors with the lowest mean squared error is 3 in this study. The table 3.5.1 and 3.5.2 show the k-value options and corresponding MSE scores while implementing XGB Regressor and Random Forest regressor.

XGB Regressor	MSE scores
3 neighbors resulted with mse:	0.10195513413480553
4 neighbors resulted with mse:	0.10283750824321
5 neighbors resulted with mse:	0.10299007041877231
6 neighbors resulted with mse:	0.10374837796871067
7 neighbors resulted with mse:	0.10342165675228852
8 neighbors resulted with mse:	0.10346832197867169
9 neighbors resulted with mse:	0.10347467690687671
10 neighbors resulted with mse:	0.10332663006818059
Best k_neighbors:	**3**
Best mse score:	**0.10195513413480553**

Table 3.5.1:k values and corresponding mse scores in XGB Regressor

Random Forest Regressor	MSE scores
3 neighbors resulted with mse:	0.1020957330099562
4 neighbors resulted with mse:	0.1027337574902651
5 neighbors resulted with mse:	0.1034560273103252
6 neighbors resulted with mse:	0.1036181535666552
7 neighbors resulted with mse:	0.10371547255665993
8 neighbors resulted with mse:	0.10332580841907263
9 neighbors resulted with mse:	0.1035581170380693
10 neighbors resulted with mse:	0.10338921916250624
Best k_neighbors:	**3**
Best mse score:	**0.1020957330099562**

Table 3.5.2: k values and corresponding mse scores in Random Forest Regressor

In both regressors, the optimal k value for oversampling is chosen as 3, regarding its' mse score. With using 3 neighbors, synthetic samples are created in the training set. The original data was split into 80% of the data as the training set and 20% as the test set. We applied the SMOTE method in the training set to reach a more balanced dataset regarding targets and test set is kept evaluating the model's performance on unseen data.

We used SMOTE package in Python to implement oversampling. Final resampled dataset sizes and resampled target values are presented in the following table.

		Oversampled Dataset (SMOTE)	Original Dataset
X_train		[113676, 14]	[66232,14]
y_train	0: Not Return	[57138,1]	[56538,1]
	1: Return	[56538,1]	[9394,1]

		Oversampled Dataset (SMOTE)	Original Dataset
X_test		[16558, 14]	[16558,14]
y_test	0: Not Return	[14210,1]	[14210,1]
	1: Return	[2348,1]	[2348,1]

Table 3.5.3:Original and Resampled Training (Left) and Test (Right) set sizes

The oversampled dataset is used in Random Forest, XGBoost, and DNN algorithms, which will be explained in detailly in the following sections.

3.5.2 Undersampling - Random Under Sampling

Undersampling is a resampling technique used to address class imbalance in machine learning datasets, mainly when one class (the majority class, Class 0: Not return in this study) is significantly overrepresented compared to the other class (the minority class Class 1: Return in this study). Undersampling involves decreasing the number of instances in the majority class to balance the class distribution.

Undersampling is typically implemented under the following circumstances:

- Significant Class Imbalance: When there is a substantial class imbalance, the majority class significantly outnumbers the minority class. This imbalance can lead to biased model predictions, as models may prioritize the majority class and perform poorly on the minority class. Undersampling helps balance the class distribution.
- Availability of Sufficient Data: Undersampling is more feasible when we have a sufficiently large dataset, especially in the majority class, so

removing some instances from the majority class does not result in a data scarcity problem.

- Prioritizing the Minority Class: When the minority class is of particular interest or importance in the problem and if we want to ensure that the model performs well in identifying instances of that class.
- Algorithm Sensitivity: Some machine learning algorithms are sensitive to class imbalance and may benefit from a more balanced dataset. Undersampling can help these algorithms perform better.
- Resource Constraints: Undersampling can be a solution when we have resource constraints, such as limited computational resources or training time, which make it challenging to work with the entire imbalanced dataset.

In this study, we chose the random under-sampling technique to obtain a balanced dataset with undersampling. Random undersampling is a straightforward resampling technique to address class imbalance in a machine-learning dataset. Random undersampling balances class distribution by randomly deleting samples from the majority class. The following steps present the algorithm stages.

- Identify the Majority Class: Begin by identifying which class is the majority class and which is the minority class in the dataset.
- Determine the Desired Class Ratio: Decide on the desired class ratio that we want to achieve after undersampling. We used a 1:1 ratio since the same ratio is applied in oversampling (equal representation of both classes).
- Randomly Select Instances: Randomly select instances from the majority class to be removed until the desired class ratio is achieved. These selected instances are discarded from the dataset.
- Resulting Dataset: After performing random undersampling, the new dataset with a more balanced class distribution is obtained. This balanced dataset can be used to train machine learning models.

While random undersampling is a straightforward technique, it does come with potential trade-offs. Removing data from the majority class may result in information loss and may only sometimes be the best approach. In some cases, it can reduce the overall dataset size, potentially reducing the model's ability to generalize effectively.

Additionally, random undersampling should be used cautiously, and its impact on model performance should be thoroughly evaluated through techniques like cross-validation.

Since we do not have a large dataset in this study, we do not expect improved performance with an undersampled dataset in ML algorithms compared to the original dataset in ML algorithms. For comparative purposes, random undersampling is implemented using the RandomUnderSampler package in Python. The resampled dataset sizes and corresponding target values are presented in the table below. The under-sampled dataset is used in Random Forest, Xgboost, and DNN algorithms to gain comparative results. Model results will be explained in the following chapters.

		Original dataset	Under Sampling
X_train		[66232,14]	[18788,14]
y_train	1: Return	[9394,1]	[9394,1]
	0: Not return	[56838,1]	[9394,1]

Table 3.5.4: Train set sizes

3.6 Predictive Models

3.6.1 Random Forest

Random Forest Regressor is an ML algorithm used for regression tasks. Its' method is called ensemble learning, and it is based on decision trees.

Random Forest Regressor is a part of the ensemble learning family of algorithms. Ensemble learning combines the predictions of multiple individual models (decision trees) to make more accurate and robust predictions.

Instead of using a single decision tree, a Random Forest Regressor creates a collection of decision trees during training. Random subsets of the data are used to train each tree, and each tree considers a random subset of features at each split. This introduces randomness and diversity into the individual trees.

When making predictions, the Random Forest Regressor collects predictions from all individual trees and typically computes the average for regression tasks. This ensemble approach helps reduce overfitting and increases the model's generalization performance.

Random Forest Algorithm Steps:

- Data Splitting: The training data is randomly divided into subsets (bootstrapped samples) with replacement. Each subset contains a portion of the original data, allowing some instances to be duplicated while others may be omitted.
- Tree Construction: Multiple decision trees are constructed independently on each of these bootstrapped samples. The randomly selected subset of features is considered for splitting at each tree node. This randomness ensures diversity among the trees.
- Prediction: When making predictions, the Random Forest Regressor collects predictions from all individual trees. For regression tasks, the final prediction is typically the average of the predictions made by each tree. This aggregation process helps reduce variance and improve the model's robustness.

Advantages of Random Forest Regressor:

- Random Forest Regressor is robust and can handle many data types and features.
- It reduces overfitting compared to single decision trees.
- It often provides accurate predictions and works well "out of the box" without extensive hyperparameter tuning.
- It can handle high-dimensional datasets and capture complex relationships in the data.

Limitations:

- Random Forests may not be as interpretable as single decision trees.
- They can become computationally expensive for massive datasets.
- The number of trees (ensemble size) and other hyperparameters may need to be tuned for optimal performance.

In this study, Random forest regressor hyperparameters are tuned with Grid Search to reach optimal performance. This process is repeated for each dataset size: original dataset, oversampled, and under-sampled cases.

To apply Grid search, the scoring metric should be appropriately set in the Random Forest related to the regression task. The R2 score is a valuable metric in regression tasks as it quantifies the goodness of fit of a model. We monitored the r2 score and implemented it as a scorer metric in grid search to find the best hyperparameters.

The R-squared (R2) score is crucial in regression tasks. It provides valuable insights into how well a regression model fits the data and explains the variance in the dependent variable.

The R2 score quantifies the ratio of the variance in the target (dependent variable) explained by the dataset's features (independent variables) in a regression model. It is a value between 0 and 1, where:

R2 score = 0 implies that the model can not explain any of the variance in the target variable.

R2score= 1 indicates that the model perfectly explains all of the variance in the target variable. It means the model's predictions are identical to the target values. Briefly, higher R2 values indicate a better fit in regression tasks.

To implement grid search in Random Forest model, parameters options and chosen ones as the best parameters monitoring r2 score, are listed below.

		Best Parameters		
Parameters	**Options**	**Original Dataset**	**SMOTE**	**RUS**
n_estimators	[10,50, 100]	100	100	50
max_depth	[1,5, 10]	5	10	5
min_samples_split	[2, 5, 10]	10	5	10

Table 3.6.1: Grid Search Parameters

3.6.2 XGBoost

XGBoost (Extreme Gradient Boosting) Regressor is a powerful machine learning algorithm used for regression tasks. It's an ensemble learning method combining the predictions of many decision trees based on gradient boosting.

XGBoost is part of the gradient-boosting family of algorithms. Gradient boosting builds a strong predictive model by sequentially combining the predictions of multiple weak learners (decision trees). XGBoost extends traditional gradient boosting by introducing several optimizations that make it highly efficient and accurate.

XGBoost uses gradient descent optimization to train decision trees iteratively. It minimizes a loss function by adding decision trees one at a time, each addressing the errors made by the previous trees. The learning rate parameter is used to control the step size during optimization.

XGBoost includes L1 (Lasso) and L2 (Ridge) regularization terms to prevent overfitting and improve model generalization. Regularization helps control the complexity of the individual decision trees.

XGBoost Regression Algorithm Steps:

- Initialization: The model starts with an initial prediction, often set as the mean of the target values in the training dataset.
- Sequential Tree Building: XGBoost builds decision trees sequentially. In each iteration, a new tree is added to the ensemble. The tree aims to correct the errors made by the existing ensemble.
- Objective Function: XGBoost uses a specific loss function to evaluate the deviation between actual and predicted values. It aims to minimize this objective function during training.
- Tree Construction: In each iteration, the model determines the structure of the new tree by finding the splits in the data (feature values) that minimize the objective function. XGBoost uses techniques like histogram-based splitting to speed up this process.
- Regularization: L1 and L2 regularization terms are applied to the new tree's structure and leaf values, helping to control the model's complexity and prevent overfitting.
- Gradient Update: After constructing the new tree, XGBoost calculates the gradient of the objective function regarding predicted values. Based on this gradient, it then updates the predictions for each instance in the training data.
- Learning Rate: The learning rate determines how much the predictions are updated during each iteration. A lower learning rate may lead to a more accurate model but requires more iterations.
- Repeat Iterations: Steps 2 to 7 are repeated for a specified number of iterations or until a convergence criterion is met.

Advantages of XGBoost Regressor:

- XGBoost is known for its high predictive accuracy and performance on various regression tasks.

- It handles missing values gracefully and can automatically handle missing data during training.
- XGBoost is efficient and scalable, making it suitable for large datasets and distributed computing.

Limitations:

- XGBoost can require careful hyperparameter tuning to achieve optimal performance.
- Training can be computationally intensive, especially with many trees and deep trees.

In summary, XGBoost Regressor is an ensemble learning algorithm that builds a strong predictive model by combining the predictions of multiple decision trees. To achieve a strong predictive model, model hyperparameters have to be tuned; the following table shows possible options in Grid Search and the best parameters chosen as the ones, monitoring r2 score.

		Best Parameters		
Parameters	**Options**	**Original Dataset**	**SMOTE**	**RUS**
n_estimators	[100, 200, 300]	100	300	300
learning_rate	[0.01, 0.1, 0.2]	0.1	0.2	0.01
max_depth	[3, 4, 5]	4	5	5
subsample	[0.8, 1.0]	1	0.8	0.8
colsample_bytree	[0.8, 1.0]	1	1	1
booster	['gbtree']	gbtree	gbtree	gbtree

Table 3.6.2: Grid Search Parameters

3.6.3 Deep Neural Network

Besides preprocessing steps, we implemented one more additional step to use as input in DNN which is user-item embedding extraction. We utilized the Bayesian Personalized Ranking (BPR) model to generate user-item embeddings. These embeddings capture the latent factors that describe the preferences and interactions between users and items. This approach allowed us to represent the relationships between users and items in a compact and meaningful way.

Bayesian Personalized Ranking (BPR)

Bayesian Personalized Ranking (BPR) is a collaborative filtering technique widely used in recommendation systems. It is specifically designed for modeling user preferences in personalized ranking tasks. BPR focuses on pairwise comparisons between items, aiming to learn the hidden factors influencing user-item interactions.

BPR is commonly employed in scenarios where the goal is to recommend a personalized ranking of items to users. BPR can include applications such as customized product recommendations in e-commerce, music or movie recommendations, and news article recommendations. BPR excels in scenarios where users' preferences are best captured by their ranking order of items rather than their explicit ratings or preferences.

Even though BPR seems like a method of matrix factorization, the critical difference between BPR and matrix factorization lies in their objectives and the nature of the data they handle. Matrix factorization aims to reconstruct the user-item rating matrix by decomposing it into low-rank matrices, whereas BPR takes a different approach by optimizing a pairwise ranking objective. Even though there are many alternative Matrix factorization-based methods to BPR, such as Singular Value Decomposition (SVD) and Alternating Least Squares (ALS), BPR is chosen for this study which needs to learn to rank items based on user preferences.

In this study, detailed ratings are not the primary focus; we aim to reach possible returns in e-commerce using user-item embeddings based on user preferences. Because

BPR is advantageous when the explicit ratings or preferences are not available or not the primary focus, but rather the order or ranking of items is of interest.

We aim to obtain latent factors and user-item embeddings from the user-item rating sub-dataset. The reason behind that can be listed as follows:

- Dimensionality reduction: User-item matrices in recommendation systems can be immensely large and sparse, making it challenging to analyze and extract meaningful patterns. Latent factors and embeddings provide a way to reduce the dimensionality of the data by representing users and items in a lower-dimensional space. This helps capture the essential characteristics and relationships between users and items more effectively.

- Understanding user preferences: Latent factors allow us to uncover users' underlying preferences and tastes. We can identify shared characteristics or attributes influencing users' choices by representing users and items in a latent space. This enables us to make personalized recommendations based on similar user preferences or item features.

- Improved recommendation accuracy: User-item embeddings provide a more nuanced representation of users and items than explicit features or ratings. By incorporating latent factors, recommendation models can capture subtle and complex patterns in user-item interactions. This leads to more accurate predictions and personalized recommendations that align with users' preferences.

- Cold start problem: Latent factors are significantly valuable for addressing the cold-start problem, where there is limited or no historical data for new users or items. By leveraging embeddings, models can make educated guesses about user preferences or item characteristics based on similarities to existing users or items in the latent space.

- Efficient computation and scalability: Latent factor models often offer computational advantages over traditional approaches. Once the latent factors or embeddings are learned, recommendations and predictions can be computed more efficiently than direct calculations on the original high-dimensional data.

In summary, latent factors and user-item embeddings are essential for this study to understand user preferences, reduce dimensionality, improve recommendation accuracy, address the cold start problem, and enable efficient computations in recommendation systems and predictive modeling tasks. They provide a way to represent users and items in a lower-dimensional space, capturing the underlying relationships and characteristics that drive user-item interactions.

Considering the above reasons, BPR is employed to reach user-item embeddings. BPR involves maximizing the posterior probability by considering the likelihood function, prior probability, and the parameter vector of a model class (such as matrix factorization). BPR formulation can be written as follows:

$$p(\Theta|>_u) \propto p(>_u|\Theta)\, p(\Theta)$$

Equation 3.6.1:BPR Formula

To apply this formulation, the user–item rating dataset should be formed like the following figure regarding user preferences. Then the interaction matrices should be obtained. In this study, the interaction matrix has 47958 rows, the number of unique users, and 1378 columns, the number of unique items.

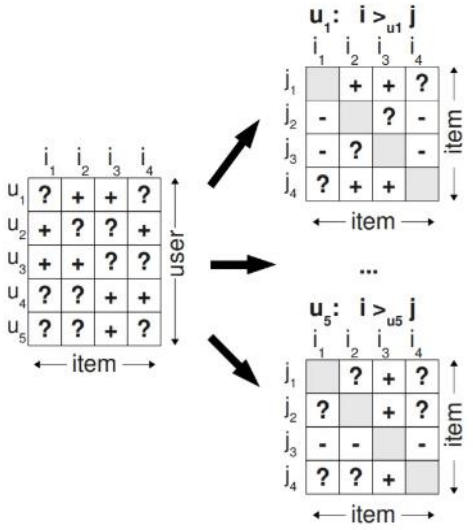


Figure 3.6.1:(u, i, j) ∈S is that user u is assumed to prefer i over j (Rendle et al., 2012)

The likelihood function represents the latent preference structure for each user individually. It captures the probability that a user prefers one item over another based on their latent preferences. The relation between user u, item i, and item j is represented by a real-valued function, typically calculated using a matrix factorization model. This function is wrapped in a sigmoid function to obtain the individual probability.

To maximize posterior probability, likelihood function and prior probability are formulated as below in previous studies (Rendle et al., 2012).

$$p(i >_u j|\Theta) := \sigma(\hat{x}_{uij}(\Theta))$$

where σ is the logistic sigmoid:

$$\sigma(x) := \frac{1}{1 + e^{-x}}$$

Equation 3.6.2: Likelihood function

The prior probability is assumed as normally distributed with zero mean and variance-covariance matrix.

$$p(\Theta) \sim N(0, \Sigma_\Theta)$$

Equation 3.6.3:Prior probability distribution

The LightFM package has been used in Python to implement Bayesian personalized ranking. After creating a user-item interaction matrix full of corresponding rating values, the LightFM model has been created with a BPR loss function.

To tune hyperparameters, grid search is implemented, finding the best parameters for user-item embeddings. Grid search is fed with multiple options. When we consider the model accuracy to tune hyperparameters, the grid search is finalized with the following best parameters.

Parameters	Options	Best Parameters
no_components	[10,20,30]	10
learning_schedule	['adagrad','adadelta']	adagrad
learning_rate	[0.01,0.001,0.05]	0.05
user_alpha	[0.0001,0.001]	0.0001
item_alpha	[0.0001,0.001]	0.0001

Table 3.6.3: BPR parameter range and best ones

User-item interaction matrix data is split into train and test sets with a 0.2 ratio. Reforming the model with the best parameters led to 60% accuracy in the test set and 66% model accuracy in the train test.

Even though it seems low accuracy to finalize with this model, it is acceptable, considering two main reasons:

1. Obtaining a good-fitted model with the following dataset characteristics is hard. The dataset must be more significant to reach acceptable accuracy, precision, and recall.

- size of dataset: (82790, 3)
- unique users: 47958
- unique items: 1378
- size of interaction matrix: (47958, 1378)

2. As explained in the method definition, BPR is a collaborative filtering method commonly used in recommendation systems. Moreover, to predict and recommend items better to users, the model needs more consecutive transactions by the same user or with the same item. Unfortunately, our dataset has mostly one transaction by the same user or with the same item. This dataset-specific characteristic harms BPR accuracy too.

In grouped dataset by item, only 382 items out of 1378 sold more than ten times. The histogram shows the total sold numbers per item on the x-axis and the count of different items with the same total sold number on the y-axis. For example, one item sold 2008 times is located on the x-axis with 2008, and the corresponding value on the y-axis is 1. Nevertheless, there is only one item having a high selling frequency. This characteristic does not hold for the whole data. The following figure shows the highest selling amounts in the dataset and the number of items sold in that frequency.

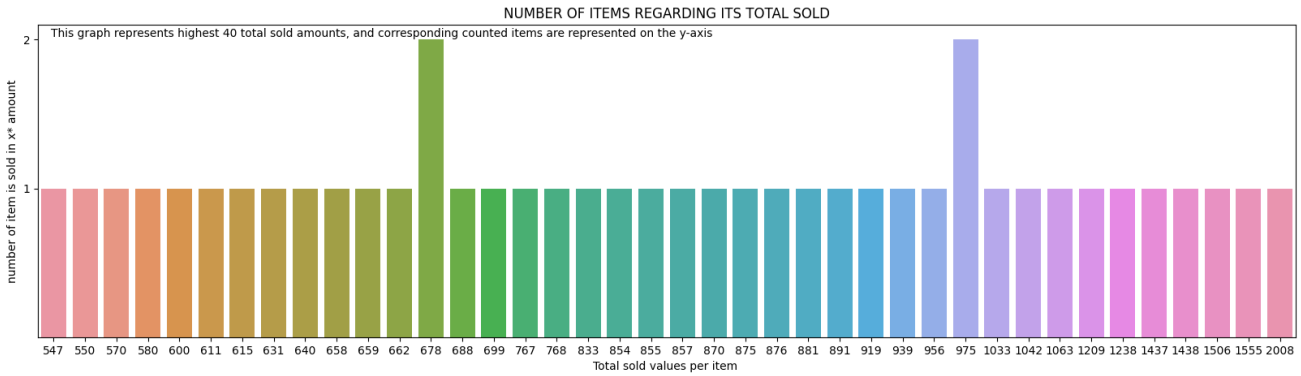


Figure 3.6.2:Number of items regarding its total sold - highest 30

On the other side, the following figure shows the dataset's lowest 30 sold values of items and corresponding counted items. Almost half of the items, 867, are sold less than 5 five times. Moreover, this characteristic significantly affects performance measurements since each item does not have enough selling frequency.

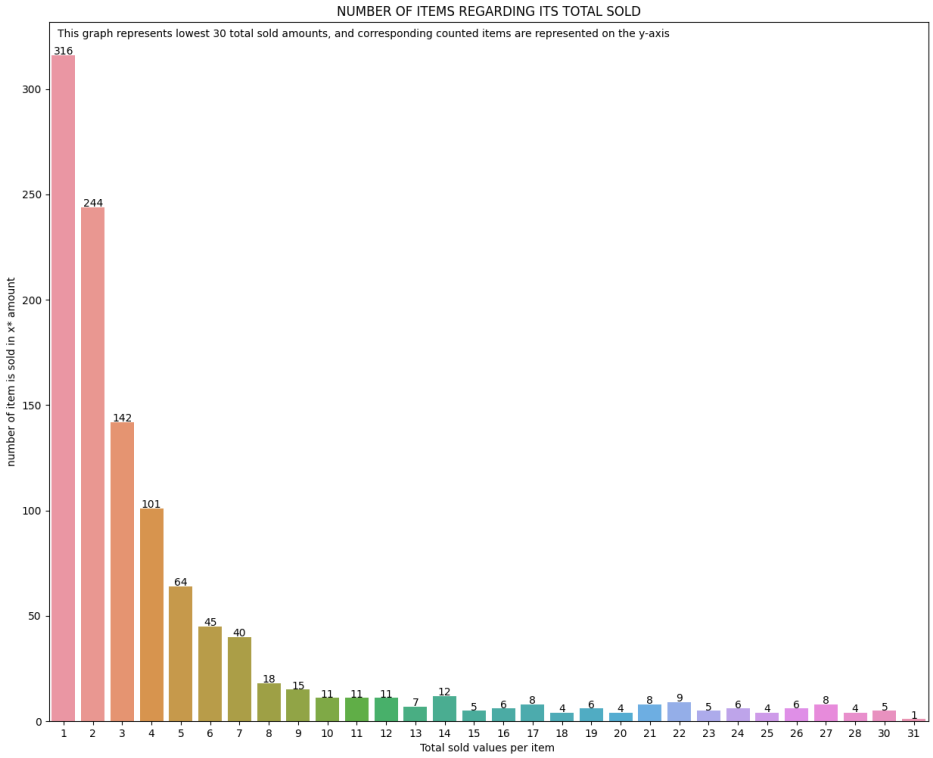


Figure 3.6.3:Number of ıtems regarding its total sold

Furthermore, the item-specific issue holds for the user purchase grouped dataset too. In the grouped dataset, considering users shows that 32029 out of 47958 users have only one purchase in the primary dataset. The following figure shows how often users purchase items and how they are placed in twenty bins.

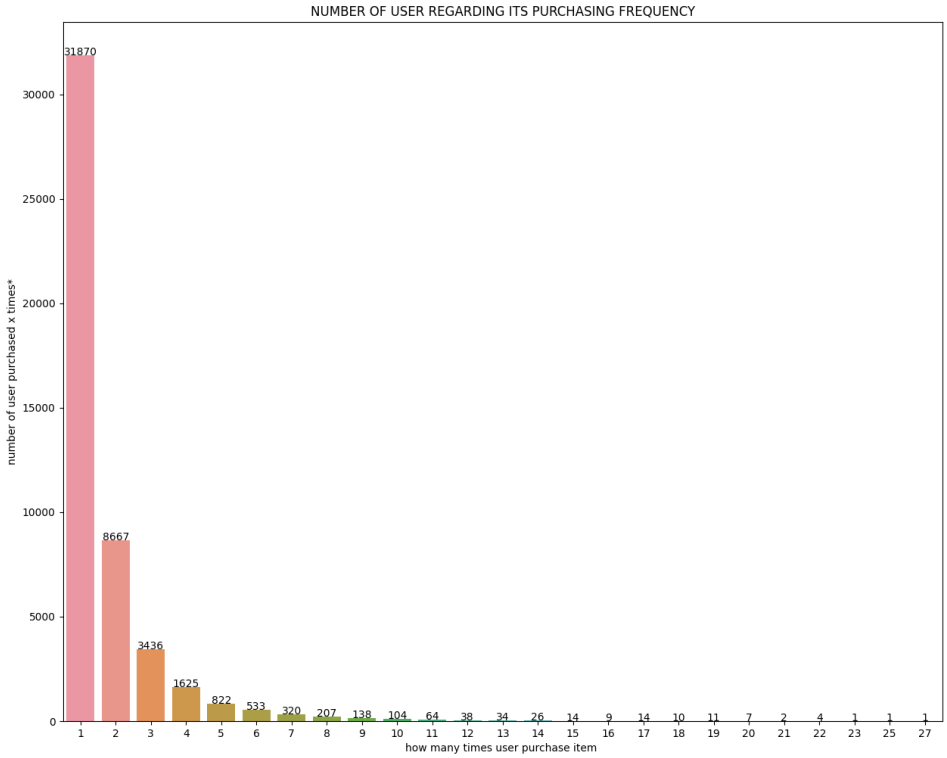


Figure 3.6.4:Number of user regarding its purchasing frequency

Considering the above two reasons, although an 80% or 90% accuracy rate is usually defined as good, we accepted a 60% accuracy rate within the scope of this study due to restrictive factors.

In addition, BPR is a commonly preferred collaborative filtering method in recommendation systems; this study employs it as a feature extraction method, like a matrix factorization method. The main difference from matrix factorization is using the BPR-specific loss function for pairwise user item ranking, which has powerful insights about users and items.

Since the study does not aim to build a recommendation system, the best feasible user item embeddings with acceptable model accuracy are sufficient. When considering the above reasons and the study's primary aim, user item embeddings from the model are assigned as one of the DNN model inputs.

Multi Input Fully Connected Deep Neural Network

A deep neural network with multiple inputs can process multiple data streams simultaneously. It consists of multiple layers of interconnected artificial neurons that process the input data and perform complex computations to generate output predictions.

Each input stream is treated as a separate layer in a multi-input, fully connected deep neural network. These input layers are typically connected to one or more hidden layers, which contain different neurons that perform nonlinear transformations on the input data. The network can learn and improve its performance by using hidden layers. Hierarchical representations and extract relevant features from the input streams.

- Every neuron in a layer must be connected to every other neuron in the subsequent layer.
- The connections between the layers are fully connected.

The system can better understand and predict the desired output by recognizing connections and correlations between inputs. It lets the network learn complex relationships and patterns between the input streams, leading to more accurate and effective results.

The output layer means producing the final predictions or classifications using the representations learned from the input streams. The network's weights and biases are updated during backpropagation to reduce errors between predicted and actual output. Multi-input fully connected deep neural networks are powerful models that can handle diverse data types from different input streams, such as text, images, numerical features, or time series data.

Deep neural networks are chosen over decision trees for several reasons in predicting returns in e-commerce. First, deep neural networks can capture complex, nonlinear relationships in the data. E-commerce data can be high-dimensional and exhibit intricate patterns and interactions among various features. With their multiple

hidden layers and nonlinear activation functions, deep neural networks can effectively model these complex relationships and capture subtle nuances in the data that decision trees may struggle to capture.

Second, deep neural networks can learn hierarchical representations of the input data. They can automatically learn and extract relevant features from the raw data, reducing the need for extensive manual feature engineering. In contrast, decision trees often rely on handcrafted features and may need help to handle large feature spaces efficiently.

Furthermore, deep neural networks are known for their scalability and ability to handle large datasets. In e-commerce, where millions of products and users may exist, deep neural networks can efficiently process and learn from sparse and large data. Decision trees, on the other hand, may suffer from performance limitations and scalability issues when dealing with such large datasets.

Lastly, deep neural networks offer flexibility in handling diverse types of data. In e-commerce, data can include various modalities such as text, images, numerical features, and more. Deep neural networks can incorporate multiple input streams and effectively process and integrate different data types, allowing for a comprehensive analysis of the e-commerce ecosystem. Considering these factors, deep neural networks provide a powerful and flexible approach for predicting returns in e-commerce, offering improved performance, better representation learning, scalability, and the ability to understand diverse data types compared to decision trees.

Considering the above reasons, a multi-input fully connected neural network is developed in this study to predict product return in the e-commerce domain. The network takes multiple inputs, including user and item embeddings, extracted features, and return labels as the target variable. The neural network architecture includes two hidden layers with sigmoid activation functions. By incorporating user and item embeddings, the network can capture the latent factors and underlying relationships between users, items, and return behavior. The extracted features provide additional information that can aid in making accurate predictions. The return labels serve as the

target variable, allowing the network to learn from labeled data and optimize its predictions. With the multi-input fully connected neural network architecture, the thesis study aims to improve the prediction accuracy of product returns in e-commerce, enabling proactive measures and optimizing supply chain management.

The multi-input fully connected deep neural network model is shown in the figure below. User-item embeddings, engineered features, and necessary columns from the dataset are utilized as inputs for DNN. The return column reached through review analysis is used as the target in DNN. The model has two hidden layers with ReLu activation and one dense output layer with sigmoid activation to have a probability-based score at the end.

In the context of Deep Neural Networks, "epoch" and "patience" are terms related to training neural networks, particularly during the process of training using techniques to prevent overfitting.

An "epoch" refers to one complete pass through the entire training dataset during the training of a neural network. In other words, an epoch is a single iteration during which the model processes all training examples once to update its weights and improve its performance.Training neural networks typically involve running multiple epochs. The number of epochs is a hyperparameter that can be specified when training the model. It determines how often the model will see and learn from the training dataset. The higher number of epochs can improve the model's performance, but it may also increase the risk of overfitting if not controlled.

To control the performance, 'patience' hyperparameters should be used as a regularization technique during training that helps prevent overfitting. Early stopping involves monitoring the model's performance on a validation dataset during training and stopping the training process when the model's performance on the validation dataset starts to degrade. The "patience" parameter is a hyperparameter that specifies the number of epochs to wait after the model's performance on the validation set stops improving before stopping the training process. For example, if we set the patience parameter to 5, the training process will continue until the model's validation

performance does not improve for five consecutive epochs. Once this condition is met, training is stopped early to prevent overfitting and store the model with the best performance on the validation set. Early stopping with patience helps balance training the model for sufficient epochs to learn functional patterns and preventing it from learning noise or overfitting the training data.

In summary, an "epoch" represents one pass through the entire training dataset during the training of a neural network, set to 50 in this study. At the same time, "patience" is a hyperparameter used in the context of early stopping to determine how many epochs to wait for improvement in validation performance before stopping the training process to prevent overfitting, which is set to 10 in this study.

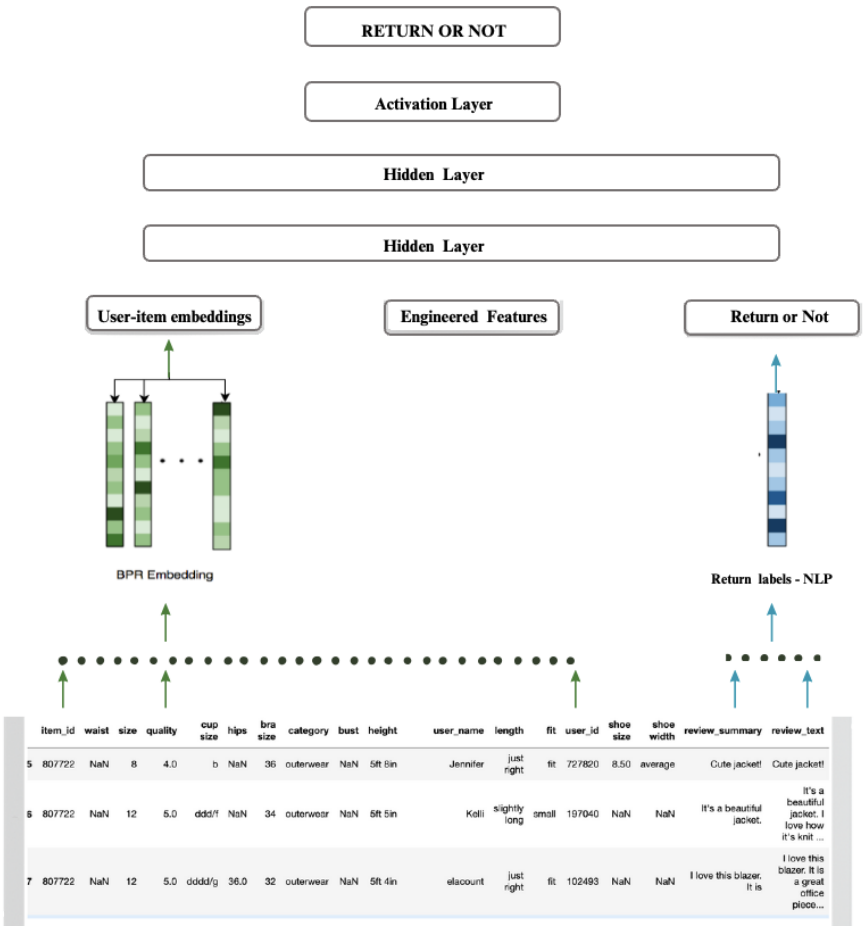


Figure 3.6.5:Model structure

The following figure presents the model architecture and the number of corresponding neurons for each layer. As explained above, neural networks are used to solve complex structures, and model architecture can be made more complex and deep in the context of the problem domain.

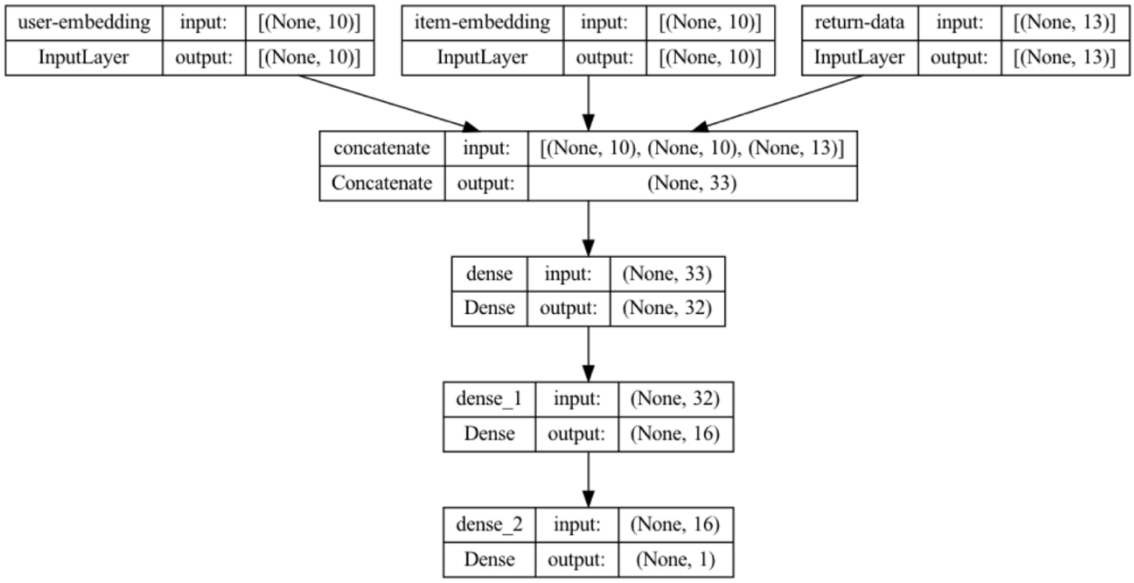


Figure 3.6.6:Model architecture

Chapter 4:

RESULTS AND ANALYSIS

This section represents the general flow of the model and its results. First, preprocessed data as input is discussed. Then, model results in each sampling cases and most important features affecting predictions are discussed. After that, comparative results are presented. At last, how the decision threshold affects the results, and its insights are discussed.

4.1 Preprocessing Steps and Predictive Approaches

In the study, we used an open-source e-commerce rating and review dataset. We implemented three different predictive approaches to obtain comparative results.

Random Forest

XGBoost

Deep Neural Network

We had to start with preprocessing due to a need for return data to implement these algorithms. Since we lacked explicit return data, we leveraged the textual information available in customer reviews related to their purchases. We applied natural language processing (NLP) techniques to analyze and extract valuable insights from the review texts. Using NLP, we could derive meaningful labels for our target variable, explicitly indicating whether a purchase would likely result in a return. The labeled return column shows that only 14% of the dataset is returned, labeled as 1 in the return column. Considering the percentages of the classes in the target column, we decided to implement resampling techniques to prevent imbalanced data sufferings. The chosen techniques are SMOTE as oversampling and Random Under Sampling (RUS) as undersampling.

As a second step in preprocessing, we performed feature engineering to replace the original rating column. This involved identifying features highly correlated with the ratings and using them as proxies to capture the underlying information. By incorporating these engineered features into our analysis, we aimed to enhance the predictive power of our model and capture more subtle patterns in the data.

Preprocessed and resampled datasets are trained with Random Forest, XGBoost, and tuning with Grid search to find the best hyperparameters for model performance.

We implemented a different approach besides the first two algorithms for the third predictive approach. We elaborated the raw dataset to capture user-item interactions in more detail. We added one more step to do it: user-item embedding extractions with BPR. This approach allowed us to incorporate additional information from customer ratings and enrich our understanding of return behavior. In this approach, we built a multi-input, fully connected deep neural network to predict returns.

To summarize, we employed three different methods with corresponding unique steps to obtain relevant insights for our study. We employed Random Forest and XGBoost with original, oversampled, and undersampled datasets to gain comparative results and insights. These two predictive algorithm results, trained with the original dataset, are used as the base score for our third approach, DNN.

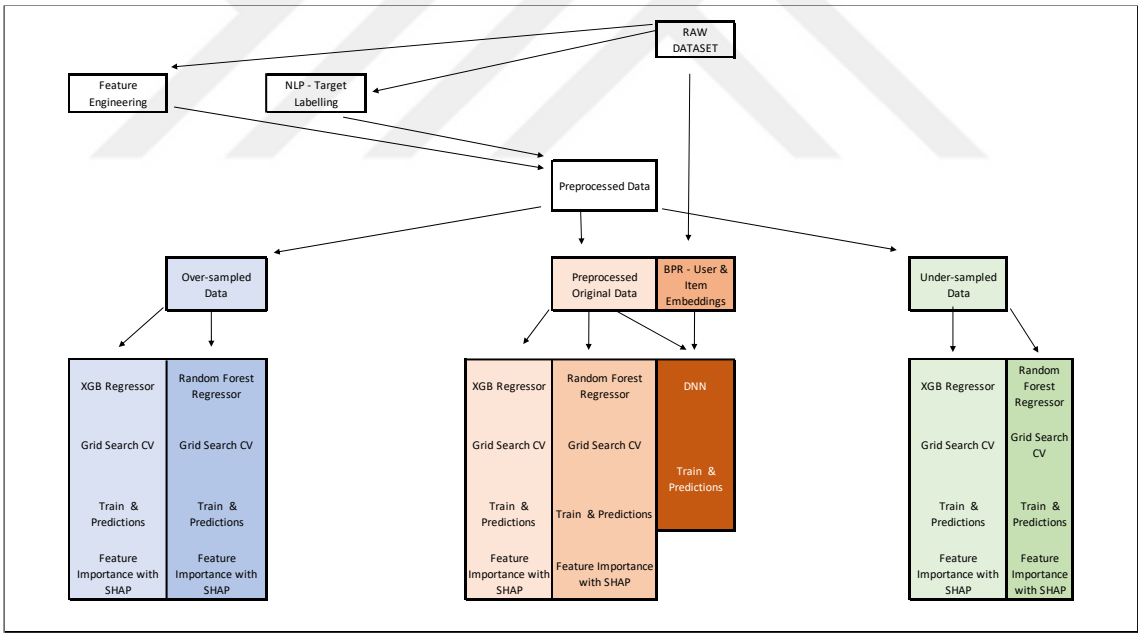


Figure 4.1.1:General Flow of Methods

4.2 Evaluation Metrics

For this study, Model accuracy, precision, and recall are evaluation metrics chosen to assess the performance of a classification model.

Model Accuracy: Model accuracy measures the overall correctness of the predictions made by the model. It is calculated as the ratio of the correctly predicted instances to the total number of instances in the dataset. Since the model has a binary classification, accuracy represents the proportion of true positive and true negative predictions out of all predictions.

The confusion matrix is described in the following figure.

		PREDICTION - Y'	
		PREDICTED NEGATIVE (-)	PREDICTED POSITIVE (+)
ACTUAL - Y	ACTUAL NEGATIVE (-)	**TRUE NEGATIVE**	**FALSE POSITIVE**
	ACTUAL POSITIVE (+)	**FALSE NEGATIVE**	**TRUE POSITIVE**

Figure 4.2.0.1 Confusion Matrix

Precision: Precision is defined as a metric that quantifies the accuracy of positive predictions. It calculates the ratio of true positive predictions to the total number of positive predictions, including both true positives and false positives. Precision focuses on the quality of positive predictions and indicates how often the model correctly identifies positive instances. In the E-commerce return prediction context, precision measures the model's ability to correctly identify and classify returned items without including items that were not returned.

A high precision value indicates that the model has a low rate of false positive predictions, meaning it accurately identifies returned items without mistakenly labeling non-returns as returns. In e-commerce return prediction, precision is an important metric as it reflects the model's reliability in identifying actual return cases. Higher precision means that the predictions made by the model are more likely to be correct, which can be valuable for firms in terms of reducing unnecessary costs associated with handling non-returns and improving operational efficiency. Precision has the following formula regarding the confusion matrix.

$$\text{Precision} = \frac{TP}{TP+FP} = \frac{P(y' = 1 \mid y = 1)}{P(y' = 1 \mid y = 1)+P(y'=1 \mid y = 0)}$$

Recall: Recall, also considered as sensitivity or true positive rate, measures the model's ability to identify positive instances correctly. Recall focuses on capturing all positive instances and is particularly relevant when the cost of missing positive cases is high. In the context of e-commerce return prediction, recall refers to the model's ability which correctly identify the returned items out of all the actual returned items. It calculates the ratio of true positive predictions (correctly predicted returns) to the total number of actual positive instances (actual returns), which includes both true positives (correctly identified returns) and false negatives (missed returns). A high recall value indicates that the model is effective at capturing and identifying the majority of the returned items. It implies that the model has a low rate of missing or failing to detect returns, which is desirable in e-commerce return prediction scenarios. By achieving a high recall, businesses can improve their ability to accurately identify and address potential return cases, leading to better customer service, inventory management, and overall operational efficiency. Recall has the following formulation.

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{P(y' = 1 \mid y = 1)}{P(y' = 1 \mid y = 1)+P(y'=0 \mid y = 1)}$$

Considering the above definitions, higher precision is much more desirable in the study's scope since the items are daily clothing, and the cost of missing or failing to detect returns is low compared to the cost of labeling non-returns as returns. Because predicting returns accurately before the item is sold and taking actions proactively regarding that prediction is the main aim of the study. However, the balance between

precision and recall metrics should always be found. The balance can shift to high recall or precision regarding the problem scope, business goals, and constraints. In this study, higher precision is preferable due to dataset characteristics.

4.3 Model Results

The analysis of the test set results revealed some interesting findings. First, approximately 14% of the data in the original set was labeled as returns, indicating a moderate prevalence of return cases in the original e-commerce dataset. In resampled datasets, training sets have a 1:1 ratio per class. The study aims to see which model and resampling technique perform better than other approaches.

As explained in previous sections, the Random Forest regressor is tuned with Grid search, and its corresponding best parameters are used for training. We used regressor instead of classification because classification thresholds may vary depending on the firms' strategies and item characteristics. Detailed threshold analysis will be discussed in further sections. We set the threshold to 0.5 and then evaluated the performance metrics. The test set is kept for prediction as unseen data.

The following table shows the test set results for each dataset in the study.

		Original Dataset	Oversampling (SMOTE)	Undersampling (RUS)
RANDOM FOREST	Accuracy	0.91	0.90	0.82
	Precision	0.94	0.74	0.43
	Recall	0.39	0.49	0.69
	F1 Score	0.55	0.59	0.53

Table 4.3.1:Random Forest Model Results

Considering the above results, we can easily conclude that the best-performing dataset is the original one in this model. In the original dataset, the model correctly classified 91% of test instances, and 94% of the predicted return cases were accurate, minimizing the occurrence of false positive predictions.

However, the model did not perform well in oversampled and under-sampled datasets compared to the original dataset.

Since there are higher number of instances in the minority class (1: return in this study) to balance the class distributions, this improves the more balanced precision and recall even they seem worse than original dataset. It leads to a better F1 score. We can conclude that the SMOTE algorithm handle with class imbalance in this study with better F1 but lower precision score. The tradeoff between evaluation metrics (recall and precision) and effects on F1 score will be discussed with more detail in further sections.

Besides oversampling, RUS perform relatively good performance to reach alike F1 score as original dataset. However, RUS involves reducing the number of instances in the majority class (0: Not return) to balance the class distributions, it leads to decrease in precision. Therefore, it improves recall for the minority class as listed in the below table. We can conclude, F1 score may not improve as much in the under sampled dataset compared to oversampled and original one.

It may be caused because of information loss during the process of resampling. Even resampling techniques are tuned to perform efficiently; these results are expected when considering our dataset size and class distributions.

While resampling techniques are valuable for addressing class imbalance, they also introduce potential challenges and trade-offs that can impact model performance. Since the original dataset is small, potential tradeoff effects dominate the model performance.

- Information Loss - leads to less representative training data and, consequently, lower model performance.
- Overfitting- may generalize poorly to the majority class instances in the test data.
- Class Imbalance Effects- leads to losing important information about the class and result in biased predictions.

Compared to the Random Forest regressor, the XGboost Regressor performs better, but the resampled datasets do not have improved performance either as explained above reasons.

		Original Dataset	Oversampling (SMOTE)	Undersampling (RUS)
XGBoost	**Accuracy**	0.91	0.90	0.83
	Precision	0.95	0.76	0.44
	Recall	0.38	0.46	0.68
	F1 Score	0.55	0.57	0.53

Table 4.3.2: XGBoost Model Results

Like the Random Forest regressor, the tuned XGBoost regressor performs better in the original dataset. It may be caused because of information loss during the process of resampling. Even resampling techniques are tuned to perform efficiently; these results are expected when considering our dataset size and class distributions.

Both regressors outperformed the original dataset, so we built a DNN structure using the original dataset. The DNN model is set to run 50 epochs with ten patience, monitoring validation recall metric. The model stopped at the 15th epoch. The train set metric history provides valuable information about the model's training progress, performance, and potential issues. It serves as a guide for making informed decisions during training and helps us evaluate the model's ability to generalize to unseen data. The following figures show the training model process epoch by epoch and corresponding model loss, precision, and recall scores for the training and validation set.

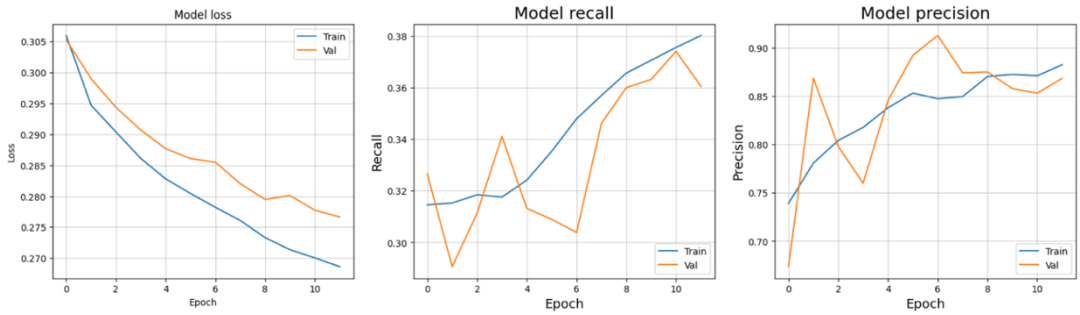


Figure 4.3.1:Model result history for training and validation set

In terms of accuracy, the model achieved an impressive accuracy rate of 89% on the test set. This indicates that the model correctly classified 89% of the instances, both returns and non-returns, which is a promising outcome. However, more than accuracy is needed to provide a complete picture of the model's performance.

The precision of the model was found to be 87% in the test set. Precision represents the proportion of predicted return cases that were indeed returned. A precision of 87% indicates that 90% of the predicted return cases were accurate, minimizing the occurrence of false positive predictions. This is particularly important in e-commerce returns, as it helps reduce the costs associated with processing unnecessary returns.

As stated in the evaluation metrics part, precision should be interpreted with recall as complementary metrics. While precision focuses on the accuracy of positive predictions, recall measures the proportion of actual positive cases the model correctly identified. Our model achieved a 31% recall score. A recall of 31% indicates that the model captured 30% of the total return cases in the test set. While this may appear relatively low, the model can still identify a significant portion of the actual return cases. A trade-off often characterizes the relationship between recall and precision. The trade-off mainly depends on the decision threshold, which will be discussed in the threshold analysis part.

As a result of analysis, the original dataset is better performed in training due to size and data characteristics than the resampled dataset in ensemble learning techniques. When we consider the performance of Random Forest, XGboost, and DNN, all these approaches performed well in predicting returns in the e-commerce dataset. The following table shows the comparative test set results of the performance metric for each approach.

Compared to Random Forest, XGBoost performs better in terms of metric scores and fits better considering r2 score. The random forest model has a 0.4 r2 score, while the XGBoost regressor has 0.42 r2 score. Besides that, the third approach showed a 0.32 r2 score, comparatively close to regressors.

		Random Forest Regressor	XGBoost Regressor	Multi-Input Fully Connected Deep Neural Network
ORIGINAL DATASET	Accuracy	0.91	0.91	0.89
	Precision	0.94	0.95	0.87
	Recall	0.39	0.38	0.31
	F1 Score	0.55	0.55	0.45

Table 4.3.3:Comparison of Model Results, training original dataset

Considering the above results and data limitations, all these models outperform to detect most of the returns even though the returns are only 14% of the dataset.

Further details and corresponding graphs are listed in the Appendix.

4.4 Feature Importance Analysis

Feature importance analysis, including techniques like stepwise feature selection and SHAP (Shapley Additive explanations) values, is essential for gaining deeper insights into machine learning models. It helps to understand which features or variables significantly impact predictions, aiding in model interpretability and decision-making. SHAP values offer a comprehensive view of feature importance by quantifying how each feature contributes to a specific prediction. This interpretability is invaluable, especially in complex models like tree-based ensembles and neural networks, as it allows us to understand not only which features matter but also the direction and magnitude of their impact.

We employed feature importance on the trained models to better understand the model's decisions using the Python SHAP package.

SHAP values indicate the contribution of each feature to each prediction relative to a baseline value (often the mean prediction). Positive SHAP values indicate that the feature pushed the prediction higher, while negative values indicate the opposite.

The summary plot and individual SHAP value plots can provide insights into which features drive predictions higher or lower and how much they contribute.

CASE I – Oversampling

The following figures show the feature importance of ensemble learning methods in over-sampled datasets. The figure on the left presents the feature importance of Random Forest, while the right one shows the feature importance of XGBoost Regressor. Even though the first three most significant features are the same in both models, the importance weights are different. In the XGBoost model, we can see more features affecting the predictions than Random Forest.

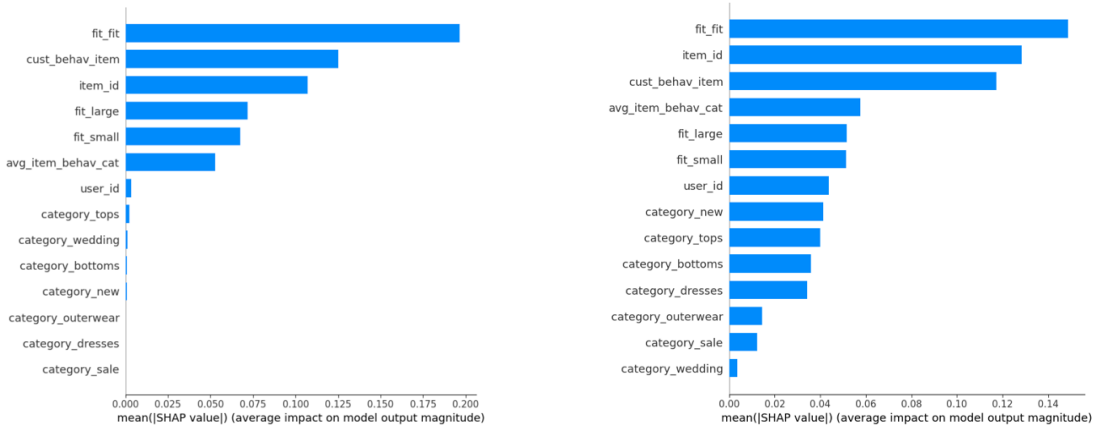


Figure 4.4.1:Shap values of Random Forest and XGBoost in the oversampled dataset

Furthermore, the most significant feature is ‘fit_fit’, a binary column filled with 0 and 1. 0 indicates the item is not a good fit for the customer, and 1 indicates the reversed scenario.

It means that fit value highly affects the return prediction as a return or not return, which is a realistic outcome. Higher SHAP values for fit_fit, cust_behav_item, and item_id make sense for the scope of this study and models.

CASE II – Undersampling

The following figure presents the feature importance of ensemble learning models in the under-sampled dataset. The left figure stands for Random Forest, while the right figure shows the important features of XGBoost model. The most significant three features are same in both regressor which are ‘cust_behav_item’, standing for customer rating behavior, ‘item_id’, and ‘fit_fit’. These models have different importance

weights on features compared to models using oversampled data. However, the fit_fit feature is still significant in decision-making in classification as a return or not.

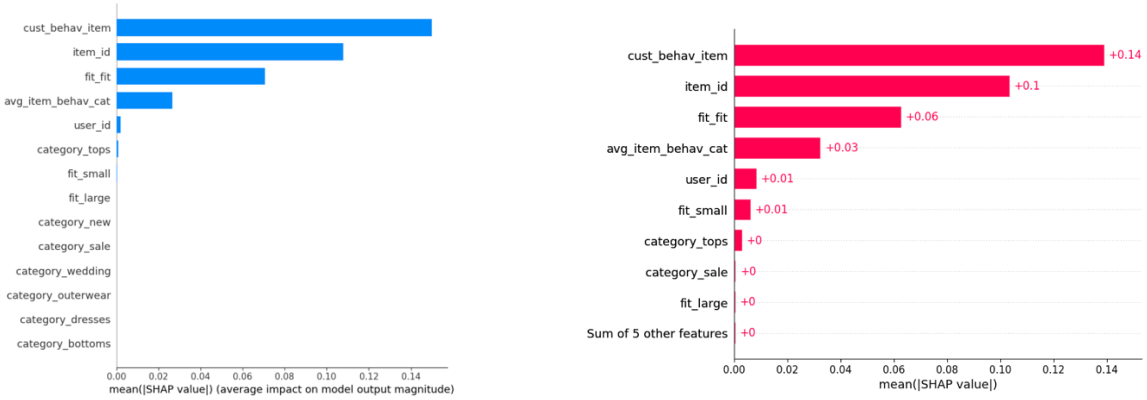


Figure 4.4.2:Shap values of Random Forest and XGBoost in the undersampled dataset

CASE III – Original Dataset

The third case is to use the original dataset for all regressors models: Random Forest, XGBoost. According to the mean Shap values of regressors, XGBoost has more diverse important features than the Random Forest model. The most important feature is specified as 'item_id' for both regressors. Models gain information regarding item_id; since we do not have a large dataset, we did not mask the item id to reach logical results. For larger datasets, it may cause data leakage; this choice should be made regarding data and problem characteristics. The second and third most important features are the same for both regressors. Customer rating behavior and fit column affect mostly regressor prediction as expected in real-world cases. However, important information should be noted while interpreting the Shap values of the model. SHAP values are not a sign of causal inference; these values do not explain how features contributed to the target variable. These values should be used to understand how each model feature has contributed to a prediction.

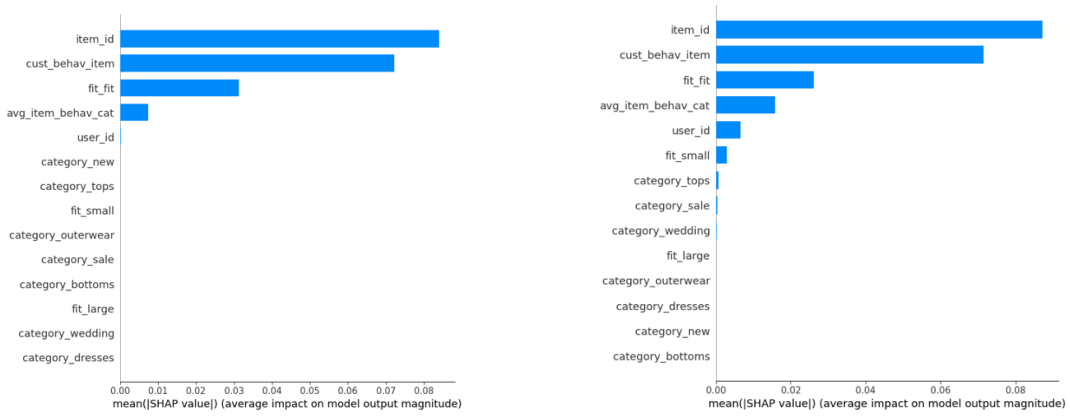


Figure 4.4.3:Shap values of Random Forest and XGBoost in the original dataset

As a drawback, feature importance analysis in deep neural networks (DNNs) can be challenging for several reasons:

- Complexity and Non-Linearity: DNNs are highly non-linear models with complex architectures containing many layers and parameters. Understanding how each feature influences the model's output is difficult due to the intricate interactions and transformations within the network.

- Black Box Nature: DNNs are often considered "black box" models because they lack transparency. They provide little insight into why they make specific predictions. This makes it challenging to interpret the importance of individual features, especially when there are many hidden layers.

- High-Dimensional Data: DNNs are commonly used for high-dimensional data, such as images and text, where feature importance analysis is less intuitive than traditional tabular data.

- Vanishing and Exploding Gradients: During training, gradients can vanish or explode as they propagate backward through deep networks. This can make it challenging to understand how features impact gradients and activations.

- Overlapping and Redundant Features: In high-dimensional data, features can be highly correlated, overlapping, or redundant, making it challenging to attribute importance to individual features accurately.

- Normalization and Scaling: DNNs often require feature scaling and normalization, affecting the interpretation of feature importance. Rescaling or transforming features can change their perceived importance.

- Interactions and Context Dependency: Features in DNNs often have context-dependent interactions. The importance of a feature may vary depending on the values of other features, making it difficult to provide a simple ranking of importance.

Considering the above reasons, employing feature importance analysis in DNN is complex and computationally expensive. Even SHAP package extensions have some attributes to analyze features in Deep Network. It is computationally expensive and complex to interpret since we have embeddings driven from BPR as input in DNN. Since the DNN structure is not basic and not sequential, importance analysis is hard to obtain to a meaningful extent. So, we compare predictive models considering their performance metrics and good fit ratios, explained in previous sections.

4.5 Threshold Analysis

Changing the binary classification threshold impacts recall and precision, affecting the F1 score. The F1 score is an evaluation metric to measure performance of a binary classification model. The F1 score combines precision and recall into a single metric, which balances the trade-off between precision and recall:

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

The F1 score helps find a balance between precision and recall. It is precious when the class distribution is imbalanced or when false positives and negatives are costly. The binary classification threshold determines the probability threshold above which an instance is classified as positive (e.g., a return) or negative (e.g., not a return). In default, the threshold is set to 0.5, and the corresponding result of this threshold is stated in the previous section. We did not change the threshold for this study since there were no business insights or any information about firms' specific desired actions. However, the threshold should be changed according to the firms' business logic. The threshold should be increased if the firm desires to identify items or users with high return probability. It leads to high precision. So, the model becomes more stringent, resulting in higher precision. A higher precision means the model is more conservative in predicting positive instances, and the predicted returns are more likely to be actual returns. However, this also leads to a lower recall because the model may miss some true positive instances, classifying them as negative.

On the other hand, lowering the threshold makes the model more lenient in classifying instances as positive, increasing the recall. The model becomes more sensitive to capturing true positive instances, including a larger proportion of actual returns in the predicted returns. However, this may come at the cost of lower precision as the model may also include more false positive instances, classifying non-returns as returns. This scenario should be employed if the cost of missing or failing return is much higher.

Therefore, when evaluating a binary classification model using the F1 score, it is essential to consider the most appropriate threshold for the specific use case and requirements. The effect on the F1 score depends on the balance between these two

metrics. If the precision improvement is substantial, it can lead to an increase in the F1 score, but it's essential to consider the trade-off between precision and recall.

Therefore, adjusting the binary classification threshold is a trade-off between recall and precision. A higher threshold emphasizes precision by reducing false positives but may sacrifice recall by potentially missing some true positives. A lower threshold prioritizes recall by capturing more true positives but may decrease precision due to more false positives. The explained trade-off can be seen in the figure below.

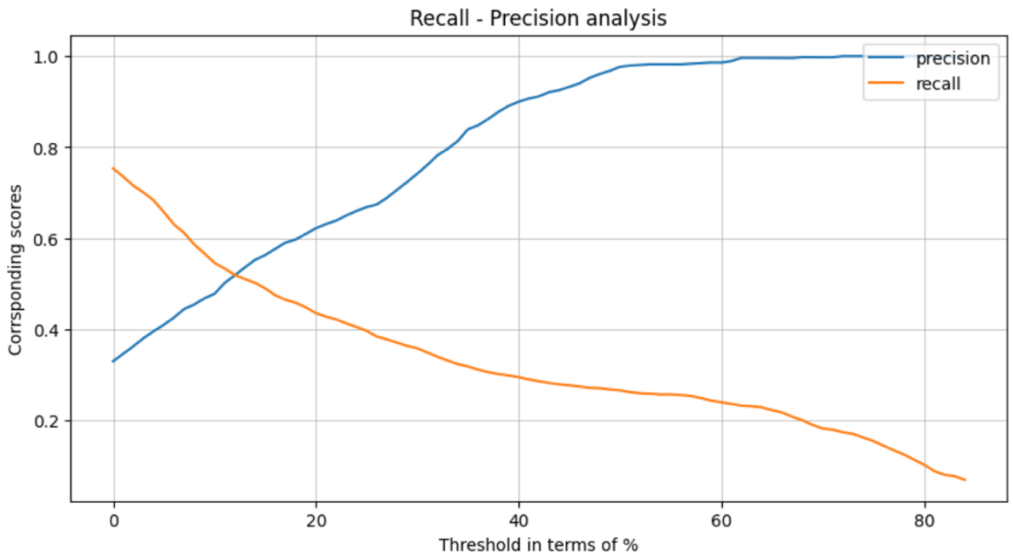


Figure 4.5.1: Precision - Recall Trade-off

Choosing the appropriate binary classification threshold depends on the specific purpose and requirements of the application. For example, if the cost of false positives (misclassifying non-returns as returns) is high, a higher threshold with higher precision may be preferred. Alternatively, if missing actual returns is more costly than false positives, prioritizing higher recall with a lower threshold may be preferable.

Ultimately, the binary classification threshold should be decided based on careful consideration of the trade-off between recall and precision and the specific needs of the firms' goals and business strategies. If the proper strategy is followed, proactive findings on return prediction will be beneficial to reduce shipping, handling, and reselling costs, besides ensuring customer satisfaction.

Chapter 5:
CONCLUSION

In conclusion, the thesis presents a comparative study for return detection, enabling firms to address return-related challenges proactively. The study involves three different methods; Random Forest, XGboost, and multi-input fully connected deep neural network. We utilize open-source data to obtain user-item embeddings, providing valuable insights into user tastes and item features hidden in latent factors. Matrix factorization based on Bayesian Personalized Ranking (BPR) is employed to generate these embeddings to use as input for DNN. We used feature extraction techniques besides the embeddings to capture the data characteristics effectively. Since no return data is available, we leverage user feedback through NLP techniques to label returns. These components are integrated into a multi-input, fully connected deep neural network. Even though the model is good enough to predict returns, our study has certain limitations. This lack of visibility into the return processes and systems of specific e-commerce platforms or retailers further complicates data access.

Besides employing DNN, we used ensemble learning methods to compare the model performances. We used engineered features and return columns derived from customer reviews to deploy regressor methods. Furthermore, regressor methods are re-built with resampling techniques to handle with imbalanced data drawbacks. Dataset size, it's characteristics highly affect the model performances even hyperparameters are tuned to reach optimal efficiency.

Since firms are not eager to share their sales and return data, we used open-source user-item rating & feedback data. The use of open-source data and the absence of a dedicated return column may limit the generalizability of our findings.

Additionally, the absence of information from the company where the data was collected restricts deciding the proper threshold. Determining the threshold should align with business goals and strategies. Since the dataset primarily consists of clothing items, a threshold of 0.5 was set, and the precision-recall tradeoff was interpreted accordingly.

The study showed that resampled datasets did not improve the model's performance too much. However, tuned ensemble learning methods outperformed as much as Deep Neural Networks. The model decision among these three approaches to deploy depends on the dataset characteristics and size. For this study, the performing model is XGboost regressor. However, in larger datasets filled with consecutive sales by the same user or with the same item, the chosen model would be DNN since the DNN model uses BPR to capture the hidden factor behind user-item interactions. The model and resampling method choice may vary regarding the dataset.

As a result of the study, comparative results present that the best-performing model is XGBoost, with minimal improvement compared to the other two models. However, the most critical features set are similar; we can point out that the ‘fit' information given by the user and the customer rating behavior derived by the user-item rating are essential to affect return predictions.

For future research, it would be beneficial to work with a larger dataset obtained directly from a company, which would enhance model choice and metrics and assist firms in devising proactive strategies. Examples of such strategies could include incurring additional shipping costs when the return rate exceeds a firm-specific threshold or implementing a customized recommendation system that considers users' preferences to prevent the occurrence of shopping for the same item in different sizes.

Overall, our study provides valuable insights into return detection and proactive measures, but further research and access to more comprehensive and specific datasets.

BIBLIOGRAPHY

Chang, Qun-Qun, and Hong-Zhen Zheng. "An Effective Strategy for Non-Defective Reverse Logistics." In 2014 IEEE International Conference on Information and Automation (ICIA), 1273–77. Hailar, Hulun Buir, China: IEEE, 2014. https://doi.org/10.1109/ICInfA.2014.6932844.

Das, D., R. Kumar, and M.K Rajak. "Designing a Reverse Logistics Network for an E-Commerce Firm: A Case Study" 13 (2020): 48–63. https://doi.org/10.31387/OSCM0400252.

Dennis, S. "The Ticking Time Bomb Of E-Commerce Returns," 2018. https://www.forbes.com/sites/stevendennis/2018/02/14/the-ticking-time-bomb-of-e-commerce-returns/?sh=3332c5394c7f.

Dutta, Pankaj, Anurag Mishra, Sachin Khandelwal, and Ibrahim Katthawala. "A Multiobjective Optimization Model for Sustainable Reverse Logistics in Indian E-Commerce Market." *Journal of Cleaner Production* 249 (March 2020): 119348. https://doi.org/10.1016/j.jclepro.2019.119348.

"Global Retail E-Commerce Market Size 2014-2023." Statista, 2021. https://www.statista.com/statistics/379046/worldwide-retail-e-commerce-sales/.

Kedia, Sajan, Manchit Madan, and Sumit Borar. "Early Bird Catches the Worm: Predicting Returns Even Before Purchase in Fashion E-Commerce." arXiv, June 28, 2019. http://arxiv.org/abs/1906.12128.

Li, Yanhui, Mengmeng Lu, and Bailing Liu. "A Two-Stage Algorithm for the Closed-Loop Location-Inventory Problem Model Considering Returns in E-Commerce." *Mathematical Problems in Engineering* 2014 (2014): 1–9. https://doi.org/10.1155/2014/260869.

Misra, Rishabh, Mengting Wan, and Julian McAuley. “Decomposing Fit Semantics for Product Size Recommendation in Metric Spaces.” In *Proceedings of the 12th ACM Conference on Recommender Systems*, 422–26. Vancouver British Columbia Canada: ACM, 2018. https://doi.org/10.1145/3240323.3240398.

Nanayakkara, Pamal R., Madushan Madhava Jayalath, Amila Thibbotuwawa, and H. Niles Perera. “A Circular Reverse Logistics Framework for Handling E-Commerce Returns.” *Cleaner Logistics and Supply Chain* 5 (December 2022): 100080. https://doi.org/10.1016/j.clscn.2022.100080.

Paksoy, Turan, and Eren Özceylan. “Supply Chain Optimisation with U-Type Assembly Line Balancing.” *International Journal of Production Research* 50, no. 18 (September 15, 2012): 5085–5105. https://doi.org/10.1080/00207543.2011.639399.

Pishvaee, Mir Saman, Masoud Rabbani, and Seyed Ali Torabi. “A Robust Optimization Approach to Closed-Loop Supply Chain Network Design under Uncertainty.” *Applied Mathematical Modelling* 35, no. 2 (February 2011): 637–49. https://doi.org/10.1016/j.apm.2010.07.013.

Ramanathan, Ramakrishnan. “An Empirical Analysis on the Influence of Risk on Relationships between Handling of Product Returns and Customer Loyalty in E-Commerce.” *International Journal of Production Economics* 130, no. 2 (April 2011): 255–61. https://doi.org/10.1016/j.ijpe.2011.01.005.

Rendle, Steffen, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. “BPR: Bayesian Personalized Ranking from Implicit Feedback.” arXiv, May 9, 2012. http://arxiv.org/abs/1205.2618.

Rogers, Dale S., Douglas M. Lambert, Keely L. Croxton, and Sebastián J. García-Dastugue. “The Returns Management Process.” *The International*

Journal of Logistics Management 13, no. 2 (July 1, 2002): 1–18. https://doi.org/10.1108/09574090210806397.

Röllecke, Felix Johannes, Arnd Huchzermeier, and David Schröder. "Returning Customers: The Hidden Strategic Opportunity of Returns Management." *California Management Review* 60, no. 2 (February 2018): 176–203. https://doi.org/10.1177/0008125617741125.

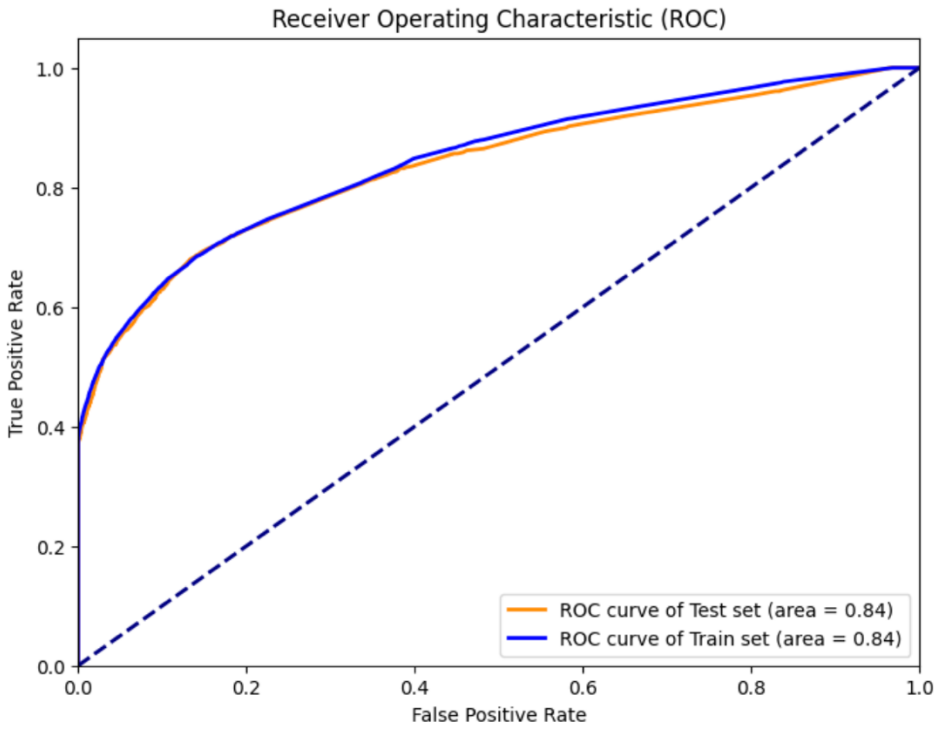
Saleh. "How-to-Create-Effective-e-Commerce-Product-Pages-Infographic." Www.invespcro.com., 2022. https://www.invespcro.com/blog/how-to-create-effective- e-commerce-product-pages-infographic/.

Toktay, Beril. *Forecasting Product Returns*. INSEAD, 2001.

Urbanke, P., J. Kranz, and L. Kolbe. "Predicting Product Returns in E-Commerce: The Contribution of Mahalanobis Feature Extraction," 2015.

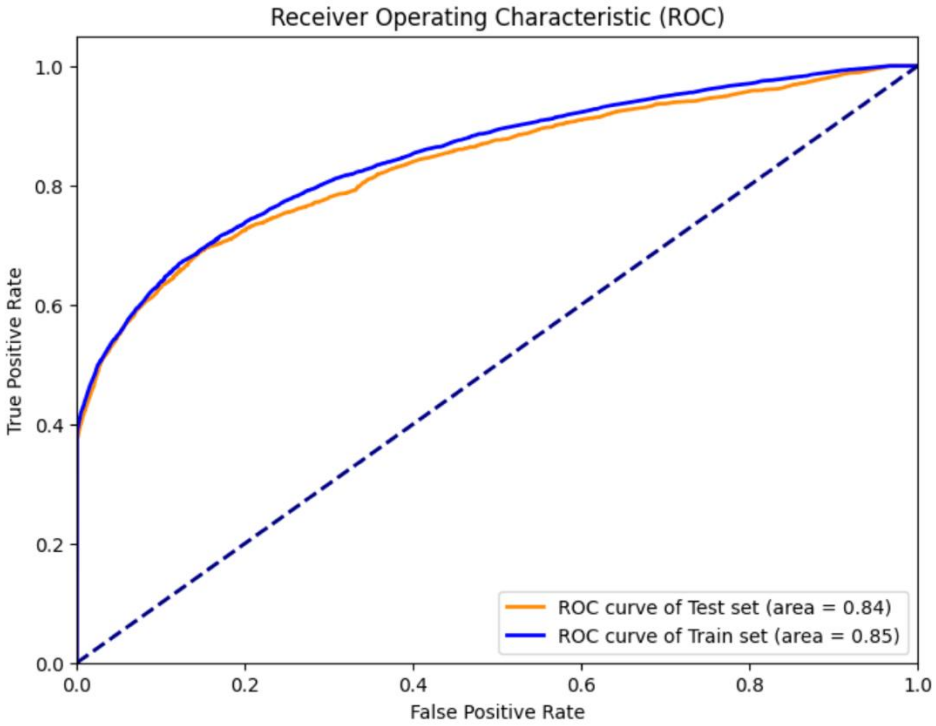
APPENDIX

Random Forest – Original Dataset



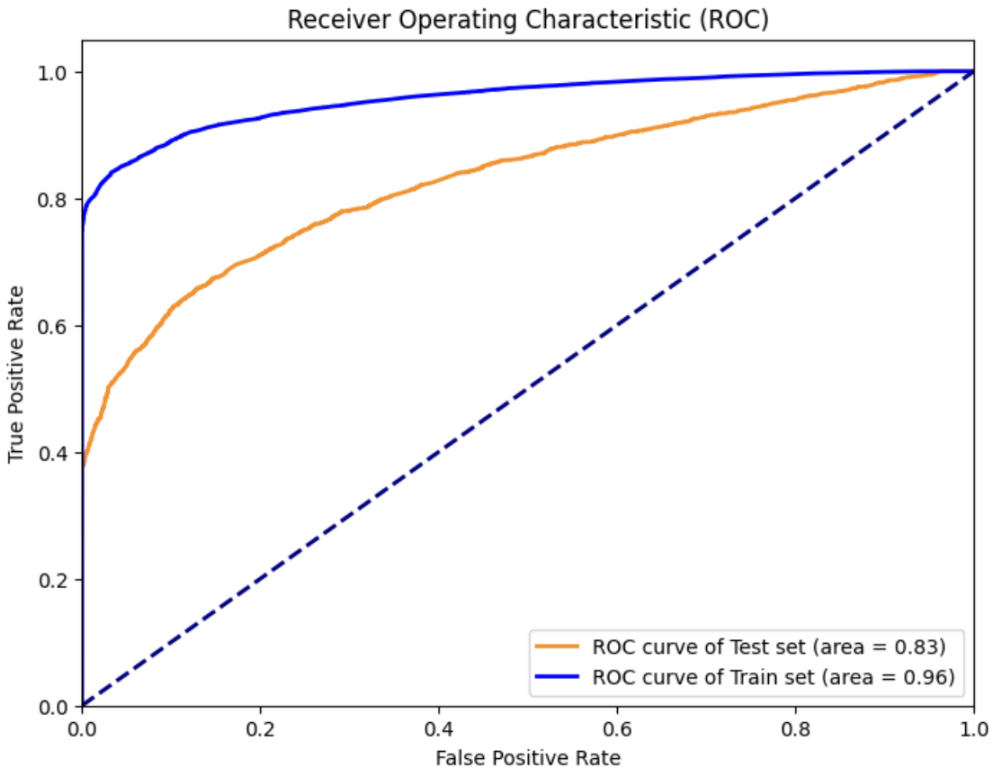
	precision	recall	f1-score	support
0: Not Return	0.91	1.00	0.95	14210
1: Return	0.94	0.40	0.56	2348
accuracy			0.91	16558
macro avg	0.93	0.70	0.75	16558
weighted avg	0.91	0.91	0.89	16558

Random Forest – RUS



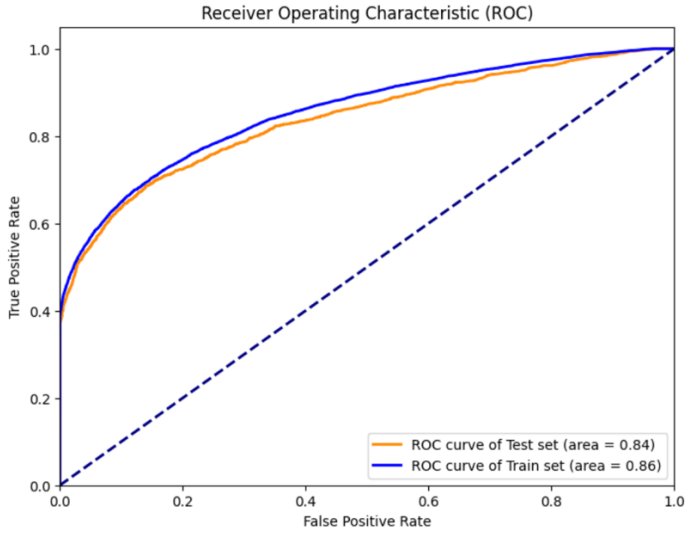
	precision	recall	f1-score	support
0: Not Return	0.94	0.85	0.89	14210
1: Return	0.43	0.69	0.53	2348
accuracy			0.83	16558
macro avg	0.69	0.77	0.71	16558
weighted avg	0.87	0.83	0.84	16558

Random Forest - SMOTE

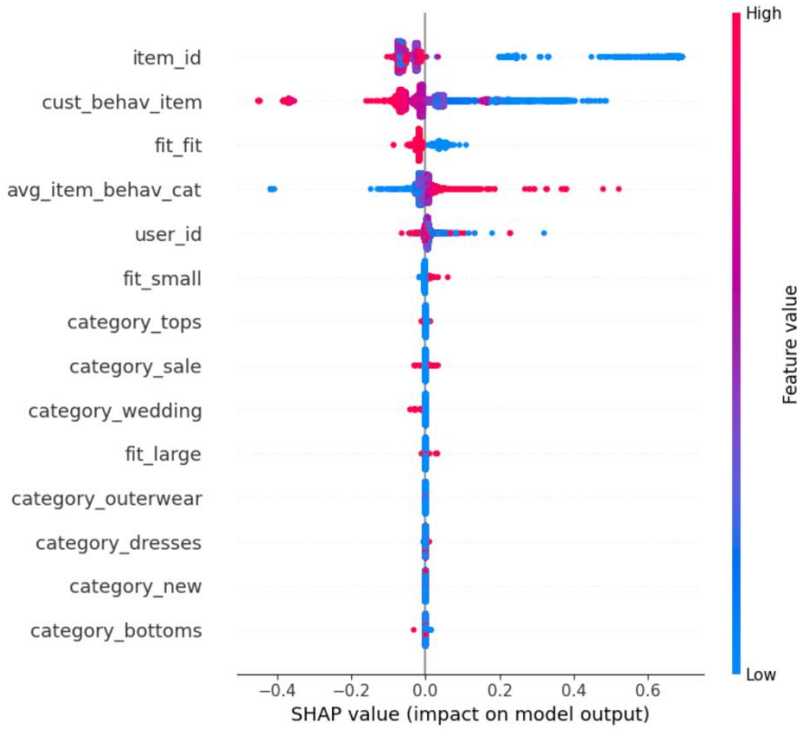


	precision	recall	f1-score	support
0: Not Return	0.92	0.97	0.95	14210
1: Return	0.74	0.49	0.59	2348
accuracy			0.90	16558
macro avg	0.83	0.73	0.77	16558
weighted avg	0.90	0.90	0.90	16558

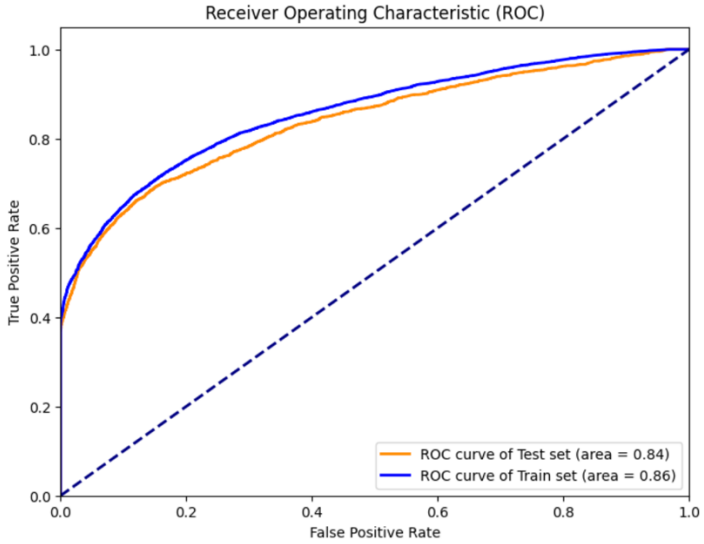
XGBOOST – Original Dataset



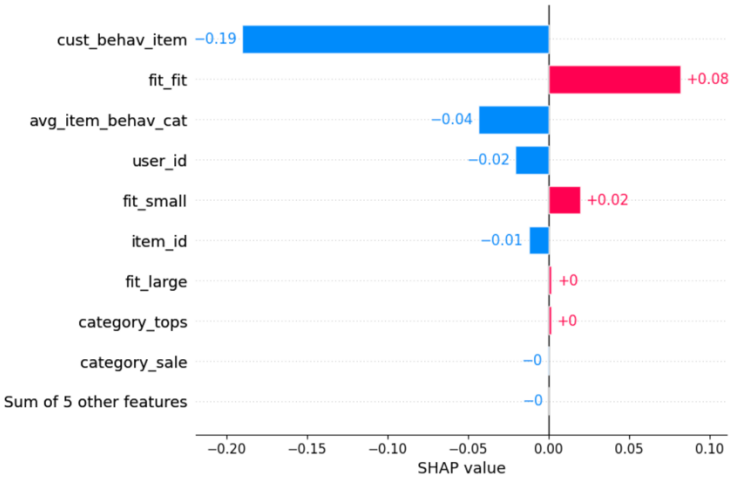
	precision	recall	f1-score	support
0: Not Return	0.91	1.00	0.95	14210
1: Return	0.95	0.39	0.55	2348
accuracy			0.91	16558
macro avg	0.93	0.69	0.75	16558
weighted avg	0.91	0.91	0.89	16558



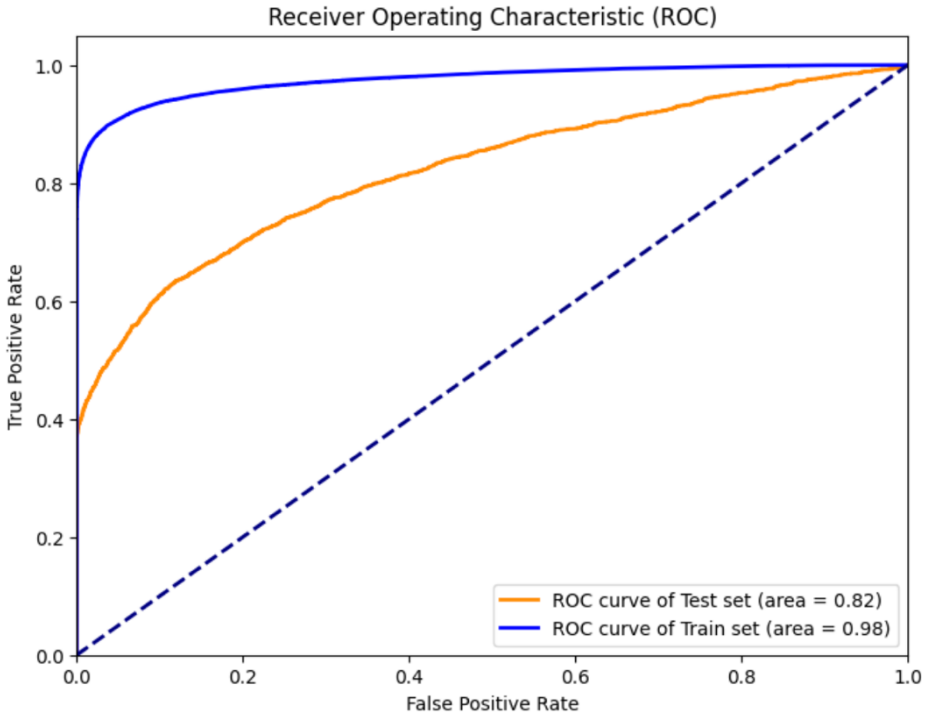
XGBOOST- RUS



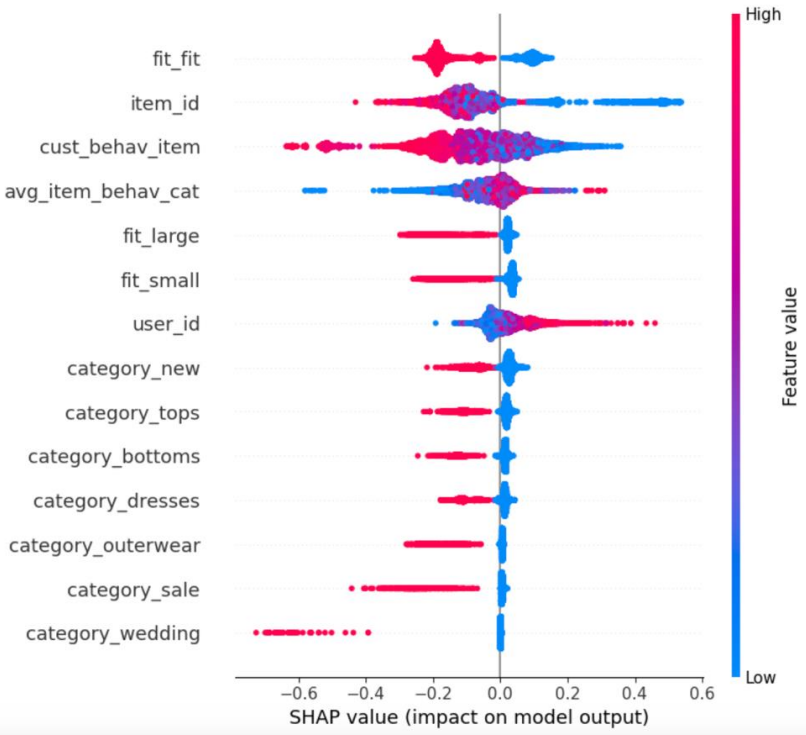
	precision	recall	f1-score	support
0: Not Return	0.94	0.86	0.90	14210
1: Return	0.44	0.68	0.54	2348
accuracy			0.83	16558
macro avg	0.69	0.77	0.72	16558
weighted avg	0.87	0.83	0.85	16558



XGBOOST- SMOTE



	precision	recall	f1-score	support
0: Not Return	0.92	0.98	0.95	14210
1: Return	0.76	0.46	0.58	2348
accuracy			0.90	16558
macro avg	0.84	0.72	0.76	16558
weighted avg	0.89	0.90	0.89	16558



DEEP NEURAL NETWORK

	precision	recall	f1-score	support
0: Not Return	0.88	0.99	0.93	14210
1: Return	0.78	0.22	0.34	2348
accuracy			0.88	16558
macro avg	0.83	0.60	0.64	16558
weighted avg	0.87	0.88	0.85	16558