

THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART
ALBERT NERKEN SCHOOL OF ENGINEERING

An Exploration of Probabilistic Model for Consumer Choices

By

Zhihao Zhang

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Engineering

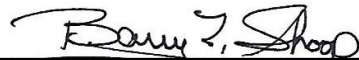
Advisor

Professor Sam Keene


THE COOPER UNION FOR THE ADVANCEMENT OF SCIENCE AND ART

ALBERT NERKEN SCHOOL OF ENGINEERING

This thesis was prepared under the direction of the Candidate's Thesis Advisor and has received approval. It was submitted to the Dean of the School of Engineering and the full Faculty, and was approved as partial fulfillment of the requirements for the degree of Master of Engineering.

 5.12.2022

Barry L. Shoop, Ph.D., P.E. - Date
Dean, Albert Nerken School of Engineering

 5.12.2022

Prof. Sam Keene - Date
Candidate's Thesis Advisor

Acknowledgments

I would like to thank Professor Keene for advising me throughout this year. This thesis would not have been possible without his help.

I would also like to thank Professor Curro for helping me understand deep learning better.

And I'd also like to thank my family for their emotional and financial support.

An Exploration of Probabilistic Model for Consumer Choices

by

Zhihao Zhang

Submitted to the Department of Electrical Engineering
on April 29, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering

Abstract

This thesis evaluates the performance of recommendation system using market basket analysis based on the SHOPPER model. The model imposes structured method to evaluate customer behavior, and takes in consideration of personal preference, price fluctuation, seasonality, as well as substitute and complement effects between products. The item-item relation is extracted based on item description as well as feature vectors generated by VGG19 from image of items. The model is compared to traditional apriori algorithm, and shows promising improvement when applied to the HM Personalized Fashion Recommendations dataset.

Thesis Supervisor: Professor Sam Keene

Contents

1	Introduction	1
2	Background	3
2.1	Machine Learning	3
2.1.1	Unsupervised Learning	3
2.2	Recommender systems	4
2.3	Market Basket Analysis	5
2.4	Associative Model	6
2.4.1	Association Rules	6
2.4.2	Apriori Algorithm	9
2.4.3	Equivalence CLAss Transformation	10
2.4.4	Frequent Pattern Growth	10
2.5	Poisson Factorization	10
2.6	Exponential Family Embedding	11
3	Previous Work	13
3.1	Shopper model	13
3.2	VGG19	16
4	Experiments	17
4.1	Data	17
4.1.1	Data Description	18
4.2	Setup	22

4.2.1	Hardware	22
4.2.2	Software	22
4.3	Implementation	23
4.3.1	Preprocessing	23
4.3.2	Baseline Model	26
4.3.3	Comparison Sets	27
5	Results and Analysis	28
5.1	Baseline	28
5.2	ELBO	30
5.3	Mean Average Precision	31
6	Conclusion	34
A	Code Blocks	35
A.1	Prepossessing	35
A.2	Shopper	38
A.3	Baseline	50
A.4	Image Feature Vector	52
B	Figures	56

List of Figures

2-1	Recommender System[6]	5
2-2	Association Rules[4]	7
3-1	SHOPPER[9]	15
3-2	VGG19 Architecture[14]	16
4-1	Example Data 1	21
4-2	Example Data 2	21
5-1	Baseline Loss	29
5-2	ELBO	30
B-1	PDF Graph	56

List of Tables

4.1	Transaction Format	18
4.2	Customer Features	19
4.3	Item Features	20
5.1	Results @ 10000 Iterations	31
5.2	Results @ 50000 Iterations	32

Chapter 1

Introduction

E-commerce markets originated as the networking between businesses[3]. With the popularization of affordable mobile device and the internet, e-commerce has evolved into a new retail channel. This new retail channel powered by modern technologies enables the retailers to present more diverse information to the customers, as well as to collect information from purchases.

While the customers are able to receive large amount of information through the platform, it is exponentially harder to quickly find the best choice. And with the ever growing popularity of online shopping, more products are made available than ever before. The mass increase of options, although can satisfy more customers, makes it ever more important to have a sophisticated recommendation system to aid the customers with decision making.

Lots of effort has been put into the development of algorithms to provide personalized recommendations for customers and improve customers' shopping experience. A common technique used to recommend items is market basket analysis (MBA)[7]. The core of market basket recommendation is building association rule sets between features, and find frequent purchase patterns in sets of items[11].

One popular approach for finding frequent patterns is to use Apriori algorithm[2], which operate under the assumption that all subsets of frequent set must be frequent.

However, traditional apriori algorithm often simplifies the decision making process and leave out many factors, such as seasonality, price fluctuation etc. And this thesis

evaluates the effectiveness of a sophisticated probabilistic model that uses a mixture of frequent set, price fluctuation, seasonal effect as well as substitute and complement relationships between items.

Chapter 2

Background

2.1 Machine Learning

Machine learning is one of the most important technologies for understanding data. And with the rapid increase in the amount of collected and generated data, it is ever more important to develop effective method to retrieve useful information from complex data[5].

Machine learning techniques are developed to find useful underlying patterns and complex relations hidden under complex data, and are capable of extracting information that humans' otherwise cannot discover. In general, machine learning can be categorized into 3 methods, supervised learning, unsupervised learning and reinforcement learning. This thesis will focus on the study of unsupervised learning and techniques that were used in the experiments.

2.1.1 Unsupervised Learning

Unsupervised learning is a category of machine learning that are used to deal with unlabeled data[18]. The technique allows algorithms to capture the patterns of datasets without human input. Unsupervised learning is used for feature selection or occasions when patterns or relations are unknown or when labeled data is unavailable.

Several popular unsupervised learning algorithms include:

1. K-mean clustering
2. K nearest neighbors
3. Principal Component Analysis
4. Association Rules

2.2 Recommender systems

Recommender systems are systems designed to filter information based on user behavior, item-relation or other underlying information to try predict the users' behavior and preference for products[17]. They are widely used in e-commerce, advertisement etc. One of the key aspect is the system personalize the recommendation with or without past records in the system. An demonstration of how recommendation system works can be seen in Figure 2-1. Lots of effort was put into studying the efficient designs of recommender systems since 1990s. In general, recommender systems can be categorized into 5 classes, including content-based, collaborative filtering (CF), demo-graphical, knowledge-based and hybrid systems.

Customers using online shopping platforms have a lot more choices than in physical retail stores because of the simplicity to present products in virtual form. This however, makes finding the ideal product exponentially harder, and as a result, efficient recommender systems are necessary to provide a good shopping experience. Targeted ads using recommender systems are also a lot more efficient since it is more likely that the viewer of the ads will like the product/service.

Collaborative filtering and association rules are some of the most popular techniques used in recommender systems. Basic descriptions of the two techniques will be provided in the later section 2.4.

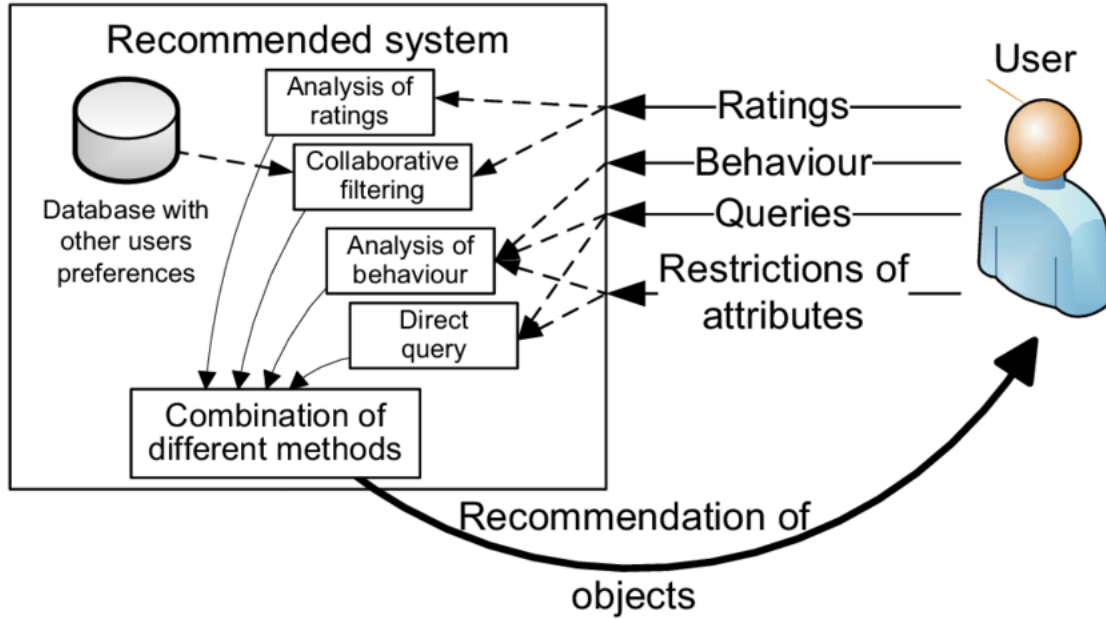


Figure 2-1: Recommender System[6]

2.3 Market Basket Analysis

Market basket analysis (MBA) is a data mining technique that mainly used to find purchase patterns from users and within products. This technique is extremely useful for decision making in retail industries, such as cross-selling, recommendations, promotions, or even the placement of items on a menu or in a store.

The core of the technique is to determine the chance of other items being considered given the current selected items [7]. The ability to predict customers' behavior in store/online market makes a big difference in sales as well as customers' shopping experience. And the power of MBA has attracted many retails enterprise such as Salesforce, Microsoft Great Plains and CRM accounting applications like Quickbooks, to have MBA as an important feature for their product[12].

Modern MBA systems often use unsupervised learning techniques such association rules mining and clustering, and can classified into two main categories, qualitative analysis and quantitative analysis. Quantitative analysis finds rules within numerical attributes, such as weight, price, and categorical attributes (color, type, etc.).

On the other hand, qualitative approach for mining association rules, as suggested by (Prakash Parvathi, 2011), mainly use binary representation of data. The binary data is then used to build decision trees, dependencies etc.

This thesis focuses on quantitative method, and uses probabilistic model to represent underlying rules between different features.

2.4 Associative Model

Associative models aim to find relationships (association rules) between provided features. These models are used to extract useful relationships in large datasets. Some of the most common use cases for associative models include market basket analysis, finding web usage patterns, detecting malicious attacks etc. And the most common techniques used for these models include association rules mining, principal component analysis, qualitative reasoning etc.

For the purpose of this paper, association rule mining will be the main technique to be discussed, since it is closely related to methods used in experiments.

2.4.1 Association Rules

Association rules[10] are set of rules that defines correlation between features. And association rule mining techniques are methods to build the rules from complex datasets. Figure 2-2 provides a visual demonstration of the process of building association rules by selecting frequent sets.

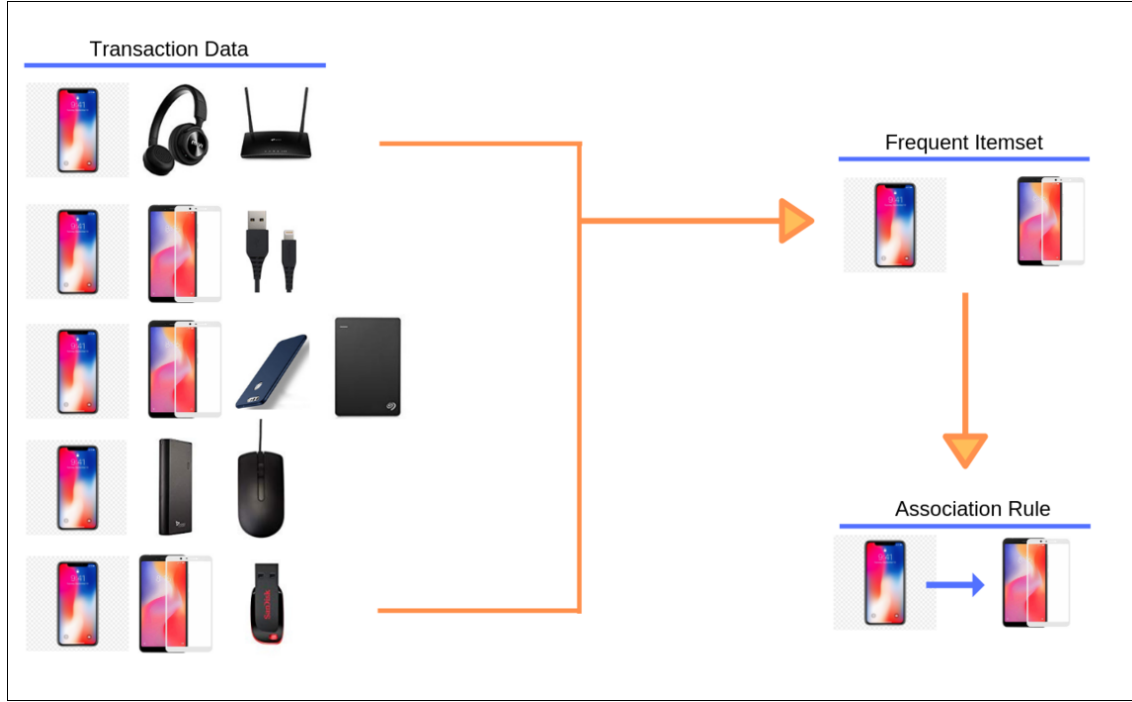


Figure 2-2: Association Rules[4]

The techniques of association rule mining was first created for knowledge discovery to efficiently handle large databases. And since these techniques came from MBA, the terminologies are defined as such:

1. Database: Collection of transactions

$$D = t_1, t_2, \dots, t_n$$

2. Transaction: Collection of items

$$t_i = i_1, i_2, \dots, i_m$$

3. Items: repeatable attributes often defined by unique id

Below is an example of the data structure:

$$D = t_1, t_2, \dots, t_5$$

$$t_1 = i_1, i_2, i_3, i_5, i_7, i_8$$

$$t_2 = i_2, i_3, i_5, i_6, i_7$$

$$t_3 = i_1, i_2, i_3, i_5, i_8$$

$$t_4 = i_1, i_4, i_5, i_8$$

$$t_5 = i_2, i_3, i_5, i_9$$

From the example above, it can be noticed that there are common patterns of occurrence between items in the 5 transactions:

$$i_1, i_3, i_8$$

$$i_2, i_3, i_5$$

$$i_2, i_3, i_5, i_7$$

.....

The frequent patterns, containing multiple association rules, can be defined as following:

$$X \Rightarrow Y$$

$$\text{where } X, Y \subseteq I \text{ and } X \cap Y = \emptyset$$

For example, the first pattern in the above list can be written as:

$$i_1 \wedge i_3 \Rightarrow i_8$$

One of the assumptions made for association rule mining is that subset of frequent sets are also frequent. And a byproduct of this assumption is that subsets of the frequent sets listed above also forms valid association rules.

For any random itemsets, the itemset is valid if the support of the set satisfies the minimum support. Support is defined as the following:

Support for itemset X: number of transactions containing X

$$\text{supp}(X) = |\{t \in T \mid X \subseteq t\}|$$

Support for association rule $X \Rightarrow Y$:

$$\text{supp}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{|T|}$$

Coverage of rule $X \Rightarrow Y$:

$$\text{coverage}(X \Rightarrow Y) = \text{supp}(X)$$

Confidence of rule $X \Rightarrow Y$:

$$\text{conf}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X)}$$

Leverage of rule $X \Rightarrow Y$:

$$\text{leverage}(X \Rightarrow Y) = \text{supp}(X \Rightarrow Y) - \text{supp}(X) * \text{supp}(Y)$$

Lift of rule $X \Rightarrow Y$:

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cup Y)}{\text{supp}(X) * \text{supp}(Y)} = \frac{\text{conf}(X \Rightarrow Y)}{\text{supp}(Y)}$$

2.4.2 Apriori Algorithm

Apriori algorithm [2] is one of the most traditional algorithms to find frequent patterns of the Boolean rules. The core of apriori algorithm is using breath-first search to generate itemsets that satisfies the minimum support requirement. The algorithm starts from itemsets of minimum number of items and use BFS to traverse to higher order. The pseudo code can be written as such:

```
L[1] = {frequent 1-itemsets};
```

```

for (k=2; L[k-1] != 0; k++) do begin
    // perform self-joining
    C[k] = getUnion(L[k-1])
    // remove pruned supersets
    C[k] = pruning(C[k])
    // get itemsets that satisfy minSup
    L[k] = getAboveMinSup(C[k], minSup)
end
return Lk (union)

```

2.4.3 Equivalence CLAss Transformation

Equivalence CLAss Transformation(ECLAT) [20][21] is a depth first search algorithm similar to Apriori algorithm. It makes use of a transaction ID (tid) list and uses this list to grow the set of itemsets by creating new itemsets from intersection of rows in tid list. ECLAT is a faster algorithm by does not provide some metrics that are available in Apriori algorithm, such as Confidence and Lift.

2.4.4 Frequent Pattern Growth

Frequent Pattern Growth technique achieves the same task by using a tree structure. It uses an extended prefix-tree structure to store the information of the frequent patterns. With this technique, it is possible to reduce dataset visits, and it adopts a pattern-fragment growth method to prevent generation of large number of sets.

2.5 Poisson Factorization

Poisson factorization[8] is a technique frequently used to represent user-item interaction by assigning a latent variable for each user and item. An example of the interaction is user preferences for items, for example a customer that has only bought vegetarian products are more likely to purchase vegetables than meat. This extracted

preference can be used for recommendation for potential purchases or can be used to make recommendation for similar customers.

The mathematical model makes the following assumptions:

1. User: user's preferences follow Gamma distribution and is reflected in past shopping activity

$$\theta_{uk} \sim \text{Gamma}(a, \xi_u).$$

2. Item: item property follows Gamma distribution and can be extracted from item popularity (purchase records)

$$\beta_{ik} \sim \text{Gamma}(c, \eta_i)$$

3. User-Item: user and item pair relationship can be expressed as Poisson distribution over user preference and item property

$$y_{ui} \sim \text{Poisson}(\theta_u^\top \beta_i)$$

2.6 Exponential Family Embedding

Exponential Family Embedding[15] is a group of method that originated from the study of natural language processing(NLP). The idea came from word embedding [13], and the concept of embedding is extended for other highly-relational datasets.

The structure of the embedding is composed of the following elements:

1. Context C_i
2. Conditional exponential family

$$x_i \mid x_{c_i} \sim \text{ExpFam}(\eta_i(x_{c_i}), t(x_i))$$

3. Embedding structure

4. Objective Function

$$L(\rho, \alpha) = \sum_{i=1}^I (\eta_i^\top t(x_i) - a(\eta_i)) + \log p(\rho) + \log p(\alpha)$$

In the context of this paper, the Context is reference to other data points (purchases). The Conditional Exponential Family are exponential probability functions that model the probability of a data point belonging to a context. The embedding is the structure of shared vectors, and in this case, each product shares 1 latent variable.

This method is commonly used for finding relationships between items, such as the similarity or the likelihood of the set of items appear in market baskets together.

Chapter 3

Previous Work

3.1 Shopper model

This thesis is largely based on the SHOPPER[16] model, which is a sequential probabilistic model for market baskets analysis.

The SHOPPER model is derived from the sequential steps that customers take when shopping, and it combines machine learning techniques with economics theory to create a structured generative model. In general, the SHOPPER model makes the assumption that customer maximize the utility of the current market basket by selecting from all the items available the one that increases the utility by the most.

The utility is modeled as a log-linear function of latent variables of items. And the latent variables includes item popularity, user preference, price sensitivity, seasonal effect and item-item relations.

The following equation describing the latent variables:

$$\psi_{tc} = \lambda_c + \theta_{ut}^T \alpha_c - \gamma_{ut}^T \beta_c \log(r_{tc}) + \delta_{wt}^T \mu_c$$

This equation calculates the utility that item c add to the trip t for the customer. The variables of the function are as follows:

1. λ_c : the popularity of item c in the dataset

2. θ_u : the user latent vector
3. α_c : item latent vector
4. $\theta_{ut}^T \alpha_c$: the relationship between user u and item c , the dot product can be viewed as the user preference for the item
5. $\gamma_{ut}^T \beta_c \log(r_{tc})$: price sensitivity of item c based on a user latent variable γ_{ut}
6. $\delta_{wt}^T \mu_c$: describes the seasonality of the item, δ_{wt}^T is time latent variable and μ_c is item latent variable

The model also takes into account of item to item interactions, including similarity, frequent occurrence in same shopping trip etc. The equation used to calculate the item-item effects is as follows:

$$\Psi(c, y_{t,i-1}) = \psi_{tc} + \rho_c^T \left(\frac{1}{i-1} \sum_{j=1}^{i-1} \alpha_{y_{tj}} \right)$$

1. ρ_c : latent variable of selected item c
2. $\rho_c^T \alpha_{c'}$: the increase in utility of buying c given c' in basket

The above equation describes the utility of a basket after adding the selected item c . The second term describes the utility of item c with consideration of all items previously selected, and the term is added to the previous total utility to measure the current total utility.

Random Utility Model is used to choose from the calculated utility:

$$\max U_{t,c}(y_{t,i-1}) = \Psi(c, y_{t,i-1}) + \epsilon_{t,c}$$

$\epsilon_{t,c}$ is the random utility, and is used to model un-considered factors in the model. The probability of customer choosing the item is:

$$p(y_{ti} = c \mid y_{t,i-1}) = \frac{\exp\{\Psi(c, y_{t,i-1})\}}{\sum_{c' \notin y_{t,i-1}} \exp\{\Psi(c', y_{t,i-1})\}}$$

And finally, the decision making step is modeled as

$$\tilde{U}_t(Y_t) = \sum_{c \in Y_t} \psi_{tc} + \frac{1}{|Y_t| - 1} \sum_{(c, c') \in Y_t \times Y_t: c' \neq c} v_{c', c}$$

The structure of the SHOPPER model can be represented as shown in Figure 3-1

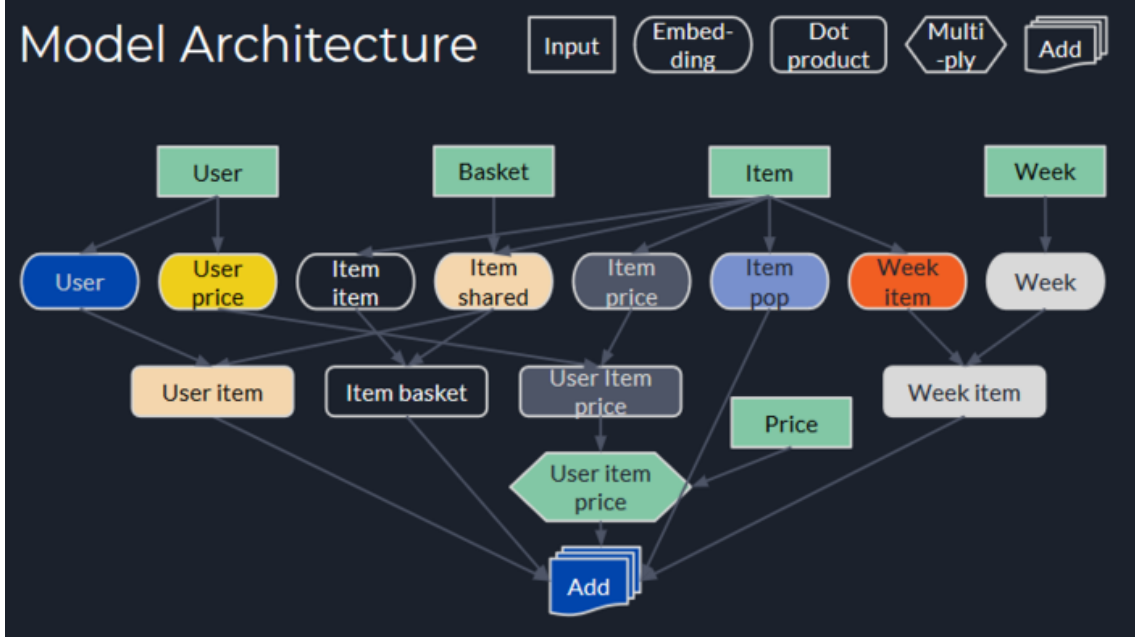


Figure 3-1: SHOPPER[9]

3.2 VGG19

VGG19[19] is a convolutional neural network that has 19 layers in total (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer), the layers configuration are shown in Figure 3-2.

The Keras model using the VGG19 architecture is trained on the ImageNet dataset and is used for image feature extraction in the scope of this paper.

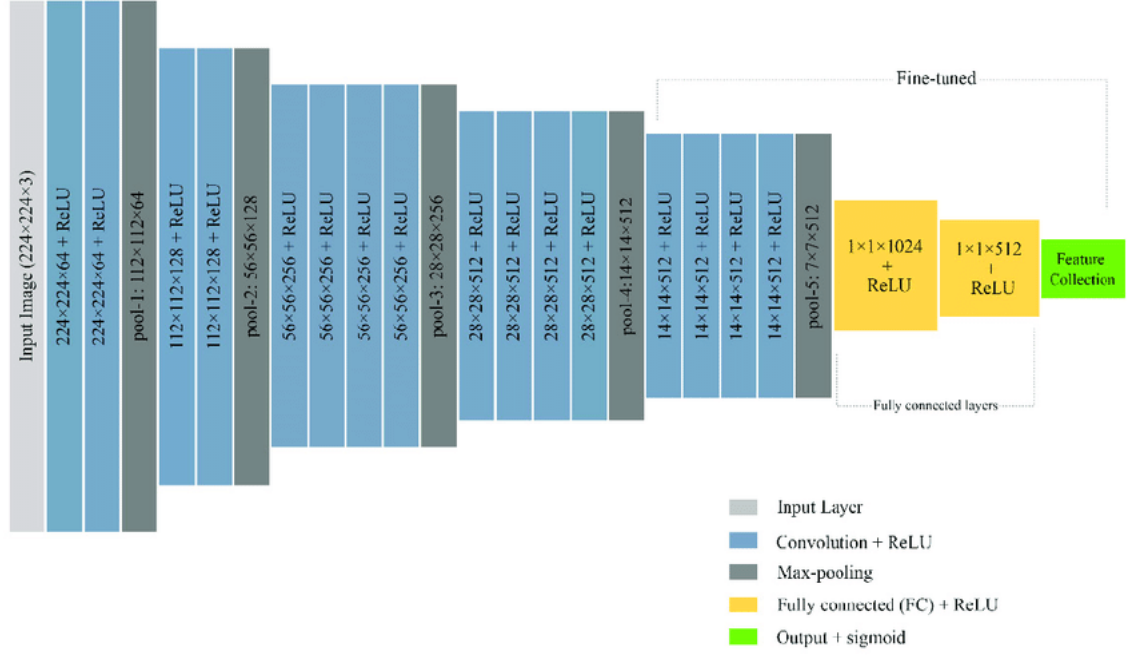


Figure 3-2: VGG19 Architecture[14]

Chapter 4

Experiments

4.1 Data

The dataset chosen for the experiment is the HM Personalized Fashion Recommendations hosted on Kaggle[1]. The HM dataset is a rather large dataset consist of close to 32 millions records of purchase, over 100k numbers of different products. And a unique feature of this dataset is that it provides image for almost each products appeared in the dataset.

4.1.1 Data Description

transactions_train.csv:

Column	Data Type	Description
t_dat	Time	Date that the transaction took place
customer_id	String	Unique identification number for customers, this id is shared in customers.csv
article_id	int	Unique identification number for products, this id is shared in article.csv
price	int	the price that corresponding product was sold at in the particular basket, the price of product is not constant throughout the time
sales_channel_id	binary	indicates whether the transaction happened on-line or in physical retail store, which one the values correspond to is not explained

Table 4.1: Transaction Format

customers.csv:

Column	Data Type	Description
customer_id	String	Unique identification number for customers, this id is shared in transactions_train.csv
FN	Binary	A column of binary data that is not explained, has over 50% missing values
Active	Binary	Active accounts that has recent purchases have this column set to 1, other wise the value is Nan, this column is useful for filtering useless account information
club_member_status	Categorical	indicates if the customer is a member of H&M, this column has other values that are not explained, and this column was never used in the experiments
fashion_news_frequency	Categorical	Categorical data that indicates whether the user receives news from H&M
age	int	age of customer
postal_code	String	hash of postal code of the user, can be used for geographical information

Table 4.2: Customer Features

articles.csv:

Column	Data Type	Description
article_id	String	Unique identification number for article, this id is shared in transactions_train.csv
product _{code}	int	the assigned unique number for the product corresponding to the name of its image in the image folder
prod _{name}	String	descriptive name of the product
product_type_no	int	the number that represents the category that the product belongs to, this is used as group for the product in baseline experiment
product_type_name	String	the name of the product type corresponding to the previous column
product_group_name	String	similar to product type but has less number of unique values
graphical_appearance_no	int	a number that is related to the appearance, no explanation is provided
graphical_appearance_no	int	a number that is related to the appearance/material of the product
graphical_appearance_name	String	name of the corresponding appearance number
colour_group_code	int	integer identifier of the color of product
colour_group_name	String	name of the color

Table 4.3: Item Features

Not all features provided for articles are listed in the table. Most irrelevant features to this experiment is left out.

image/: The images provided are formatted to similar width but the dimensions are not guaranteed to be the same, as shown the Figure 4-1 and Figure 4-2.



Figure 4-1: Example Data 1



Figure 4-2: Example Data 2

4.2 Setup

4.2.1 Hardware

The experiments in this paper were all performed on personal computer using the following hardware:

1. CPU: Intel(R) Core(TM) i9-8950HK CPU @ 2.90GHz, 2900 MHz
2. RAM: 32GM RAM
3. GPU: Nvidia GeForce GTX 1080 Mobile
4. Operating System: Arch-Linux
5. Storage: 500GB SSD

4.2.2 Software

The experiments done for this thesis involves many components. Programs used in the experiments are written in C++/CUDA and Python. And since part of the implementation is based on the original SHOPPER model, and uses pretrained model, many dependencies are used to be able to run the preexisting work.

For python environment:

1. Python 3.9
2. Numpy 1.20.3
3. Pytorch
4. Pymc3
5. Sklearn
6. Keras

For CUDA environment:

1. CUDA compiler driver (NVCC)

4.3 Implementation

4.3.1 Preprocessing

The H&M dataset is relatively complex with many features that are not necessary within the scope of this paper. Hence, only some of the provided features are selected, and a new smaller dataset is created.

The features selected to be part of the smaller dataset used for the experiments include:

1. t_dat
2. article_id
3. customer_id
4. price
5. product_type_no
6. product_code
7. image

And as mentioned previously, the original H&M dataset has over 31 millions of transactions. And if all of the available data points are used for the experiments, the training time be too long. Hence, around 5% of the original dataset was randomly selected for the experiments.

The transactions are formatted such that each transaction only contain one product. This format results in multiple rows belong to the same basket. And since "true basket" was not provided in the dataset, it is assumed that for any transactions that happen on the same day and is done by the same customer, the transactions belong to the same market basket. It is also necessary to mention that since the feature "sales_channel_id" was discarded in earlier steps, all transactions are assumed to have the same sales channel. The transactions are then grouped by the date and

customer into baskets. After the baskets are created, the baskets are then randomly selected to form the new dataset.

With the new dataset created, it is then necessary to format the dataset into the format that the original SHOPPER implementation in CUDA accepts:

Train/Test/Validation:

1. user_id
2. item_id
3. session_id
4. quantity

The quantity for each row are 1 since the dataset was expanded after grouped into baskets, and it was mentioned in the original SHOPPER documentation that the quantity is indeed ignored internally. Price:

1. item_id
2. session_id
3. price

One of the key requirements for the dataset format is that prices for each item has to be available for all sessions that appears in the transactions.

However, this rises the problem that with the number of transactions reduced, the price index became sparser. Even with the original dataset, it is clearly impossible that prices are available for any item-session combinations.

Other assumption is made so that the dataset can be formatted to meet the requirement: the price of any item is assumed to be constant if no price change is specified. This assumption allows the price to be filled by previous price of the item. However, it is also possible that no previous purchase of an item had happened at a given time, and it is necessary to assume that previous price of the item is the same as the first available price.

With the 2 assumptions made, the dataset can be formatted in the desired format.

Seasons:

1. session_id
2. week_id,
3. day_id
4. hour_id

This part of the dataset is created by getting the corresponding week number from the date of each session, and the day and hour are filled with 0 since the two fields are internally ignored.

Item Groups:

1. item_id

2. group_id

The item-group mapping is used to enforce item-item relationship such that items of the same group share certain parameters. Multiple approaches were tested to generate the item-group mapping.

4.3.2 Baseline Model

The model is developed using traditional architecture, and is intended to be used as a baseline comparison for models based on the SHOPPER model. The general structure of the model is presented in the pseudo code below, and it contains 3 convolution layers and has 1 activation layer using the LeakyReLU function between two convolution layers.

```
Baseline(  
  (article_emb): Embedding(72582, 512)  
  (top): Sequential(  
    (0): Conv1d(3, 32, kernel_size=(1,), stride=(1,))  
    (1): LeakyReLU(negative_slope=0.01)  
    (2): Conv1d(32, 8, kernel_size=(1,), stride=(1,))  
    (3): LeakyReLU(negative_slope=0.01)  
    (4): Conv1d(8, 1, kernel_size=(1,), stride=(1,))  
  )  
)
```

The data used for this model follows the same format from preprocessing so that all models are trained on the same dataset.

4.3.3 Comparison Sets

Constraints are implemented on some of the item latent variables so that items within the same group share the same latent variable, thereby changing the embedding structure in the model. The price variable βc and season variable Hlc_c are shared across groups.

Different grouping methods are tested and compared:

1. Group by item number
2. Group by Product Type
3. Group by Department
4. Group by VGG-19 Similarity (0.7)
5. Group by VGG-19 Similarity (0.55)

The VGG-19 trained model on Keras is used for extracting image feature vectors, the last two layers are removed and the extracted features are compared under cosine distance metrics. Items with similarity score over the selected threshold are grouped into same set, and each set uses the first element as representation. The sets are constructed using disjoint sets method.

The two different grouping method using VGG-19 similarity has different thresholds, the first one has higher threshold at 0.7, and the second one has 0.55.

Chapter 5

Results and Analysis

5.1 Baseline

Since the dataset chosen for the experiment (H&M dataset) is relatively new and only a smaller subset of the dataset is used for the experiment, there is no previous work that can be used for comparison. Hence, it is important that a baseline model is created and trained on the same dataset. And the performance of the baseline model needs to be studied so that a reasonable conclusion can be made for SHOPPER model.

The baseline model, as described in the earlier section, is a relatively simple model that is only composed of 3 convolutional layers with activation layer after each convolution.

The training setup of the baseline model used during the experiment has the following parameters:

1. Batch Size: 128
2. Shuffle: True
3. Iteration per Epochs: 2344
4. Epochs: 5

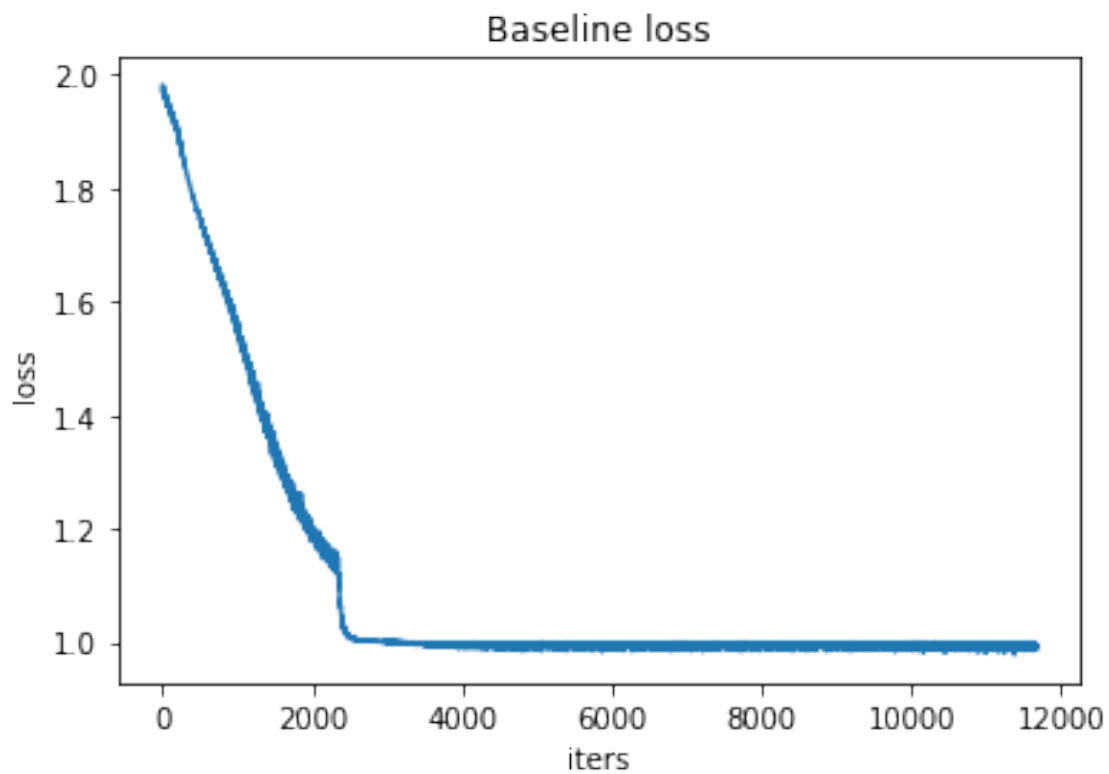


Figure 5-1: Baseline Loss

The loss during training is shown in Figure 5-1. It is quite obvious that there is a sudden drop in the loss during training, and the drop also happened quite early in the training process. This normally indicates that right after the 2000 iteration, the learning rate was reduced and the model was allowed to traverse through the weight space in more granular fashion.

5.2 ELBO

The evidence lower bound(ELBO) graph is not generated by the original SHOPPER implementation, and the python port of SHOPPER model was used to generate the ELBO graph. Given the significant training speed difference, the model was trained for smaller number of iterations.

The ELBO is a way to calculate the lower bound for the log-likelihood of observations, and as shown in the ELBO plot, the SHOPPER model using the grouping method 1 converge slowly, and this would help understand the results in the following part of the chapter.

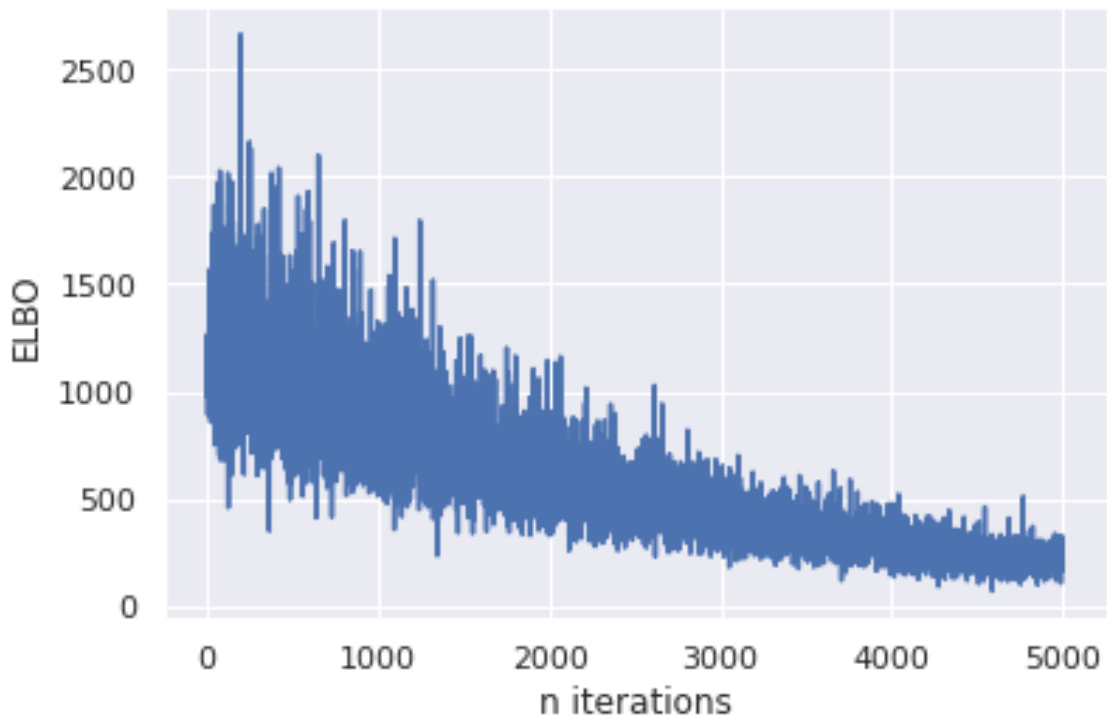


Figure 5-2: ELBO

5.3 Mean Average Precision

The task of the models are to predict the 12 items that the customer is most likely to buy in the week after the end date of the dataset. Hence, the predicted output cannot be calculated using the standard Accuracy/Precision method. The method used to score the final prediction of the models is Mean Average Precision, and with 12 entries, the score can be calculated as the following:

$$\text{MAP@12} = \frac{1}{U} \sum_{u=1}^U \frac{1}{\min(m, 12)} \sum_{k=1}^{\min(n, 12)} P(k) \times \text{rel}(k)$$

10,000 Iterations

Baseline	Item Number	Product Type	Department No	VGG-19 (0.75)	VGG-19 (0.55)
0.0207	0.0219	0.0226	0.0229	0.0221	0.02

Table 5.1: Results @ 10000 Iterations

At 10000 interactions, the MAP12 scores for all methods are shown above. It is not surprising to see that SHOPPER based model performed better than the baseline method by a decent margin. However, within all the SHOPPER based model, the one using the default feature from the original dataset "Product_Type_no" performed only marginally worse than the model using item-group derived from image feature vector.

Since from the definition of SHOPPER model, we can tell that the item group is used to enforce item-item relationship, making certain latent variables shared between different items. Using the Product type number as item group means that it is less likely that products share the same group, resulting in less strict item-item relationship, and the latent variables are capable of representing the relationships.

The VGG-19 method with threshold set at 0.75 out performs itself when threshold

set at 0.55, this also reinforces the hypothesis that large item-groups could potentially penalize model performance.

However, comparing the different method of creating item groups, the ones using provided features over performed the ones using similarity scores. The two methods using features provided in the dataset are the Product Type and the Department No columns, and the Department No group performs significantly better than the other. The product type number feature has under 800 categories while the department number has over 7000 unique values. It is suspected that too many items are sharing the same group using the product type number, which makes the price and season feature has larger error, resulting in less precision.

Comparing the VGG-19 methods to the item number grouping, which is an one-to-one map to each item and makes the model the same as original SHOPPER model without any modified structure, the similarity method with threshold at 0.75 performed better than original model. However, the same structure but with threshold 0.55 performed worse than the baseline model. An observation on the grouping method is that with threshold at 0.55, there was only 49 groups, which is significantly less than other grouping methods.

50,000 Iterations

Baseline	Item Number	Product Type	Department No	VGG-19 (0.75)	VGG-19 (0.55)
0.0211	0.0232	0.0228	0.0230	0.0229	0.0217

Table 5.2: Results @ 50000 Iterations

At 50000 iterations, the original SHOPPER with no constraints performs better than the other methods, and is able to achieve significantly higher MAP score compared to 10000 iterations. This result aligns with the ELBO plot of the model, where it can be seen that at 10000 iterations, the lower bound has not converge, indicating that more iterations is required.

The MAP score for VGG-19(0.75) is relatively close to that of using provided features(Product Type Department Number). Compared to experiments done at 10000 iterations, this method achieved decent improvement and behaved similar to the Item Number group. Similarity group with threshold at 0.55 continued to under perform but has higher precision over baseline model.

Chapter 6

Conclusion

In this thesis, we proposed utilizing pre-trained convolutional neural network to calculate similarity scores between items, when image of item is available. This further enhances the idea of embedding preexisting knowledge into the SHOPPER model. The core idea is similar to that of the original SHOPPER model, which is to combine theories in economy and machine learning techniques to achieve better performance.

Using image feature vector for item-group relation makes the model fits to the observations faster than without the shared latent variable. The grouping method is compared to other grouping methods using manually labelled features. The method using similarity score behaved similarly to the other methods. This suggests that feature extraction using trained models has embedding information similar to the manual labels, and invites more discussion to similar ideas using image feature as replacement for manual labelling when manual labels are not available. We also discovered that large item-groups could have negative impact on the model’s performance. More tuning could have been done to further optimise the threshold value, and it is also possible to combine other provided feature vectors with image similarity. However, the experiments are largely limited by the processing speed and fast storage (RAM and VRAM) of the setup.

Appendix A

Code Blocks

A.1 Prepossessing

```
class PreProcess(object):

    def __init__(self):
        self.sess = None
        self.validation = None
        self.test = None
        self.train = None
        self.groups = None
        self.codes = None
        self.uniques = None
        self.data = None
        self.transaction = None
        self.price = None
        self.baskets = None
        self.articles = None

    def load_transaction(self, transaction: pd.DataFrame, entries: int =
        10000):
```

```

# store transaction variable
self.transaction = transaction

self.transaction['price'] = self.transaction['price'].apply(lambda
    x: round(x * 1000, 2))

self.baskets = self.transaction.groupby(['t_dat', 'customer_id']) \
    .agg(lambda x: list(x))(['article_id', 'price']).reset_index()

self.baskets.index = self.baskets.index.set_names(['session_id'])

self.baskets = self.baskets.reset_index()

self.codes, self.uniques = pd.factorize(self.baskets['customer_id'])

self.baskets['customer_id'] = self.codes

self.baskets['t_dat'] = pd.to_datetime(self.baskets['t_dat'],
    infer_datetime_format=True)

self.baskets['t_dat'] = self.baskets['t_dat'].dt.week

self.data = self.baskets[['customer_id', 'article_id', 'session_id']]

self.data['quantity'] = 1

self.data = self.data.explode('article_id')

self.data['article_id'] = pd.to_numeric(self.data['article_id'])

self.price = self.baskets[['article_id', 'session_id', 'price']]

```

```

self.price = self.price.apply(lambda x: x.explode() if x.name in
                               ['article_id', 'price'] else x)

self.combo_df =
    pd.DataFrame(data=list(product(self.price['article_id'],
                                   self.price['session_id'])), columns=['article_id', 'session_id'])
self.combo_df['price'] = np.nan

self.combo_df.set_index(['article_id', 'session_id'], inplace=True)
self.combo_df =
    self.combo_df[~self.combo_df.index.duplicated(keep='first')]

# df1.set_index(['Code', 'Name'], inplace=True)

self.price = self.price.set_index(['article_id', 'session_id'])
self.price = self.price[~self.price.index.duplicated(keep='first')]

self.combo_df.update(self.price)
self.combo_df.reset_index(inplace=True)
self.combo_df['price'] = self.combo_df.groupby(['article_id'], as_index=False,
                                              sort=False).transform(lambda x: x.ffill().bfill())['price']

self.sess = self.baskets[['session_id', 't_dat']]

self.sess['day'] = 0
self.sess['hrs'] = 0

def load_articles(self, article):
    self.groups = article[['article_id', 'product_type_no']]
    self.articles = article

def split(self, train=0.7, test=0.15, validation=0.15):

```

```

self.train, rest = train_test_split(self.data, train_size=train)

self.test, self.validation = train_test_split(rest, train_size=test)

def save(self, path='./data'):
    train_path = os.path.join(path, 'train.tsv')
    test_path = os.path.join(path, 'test.tsv')
    validation_path = os.path.join(path, 'validation.tsv')

    price_path = os.path.join(path, 'item_sess_price.tsv')
    session_path = os.path.join(path, 'sess_days.tsv')
    group_path = os.path.join(path, 'itemGroup.tsv')

    self.train.to_csv(train_path, sep="\t", index=False, header=False)
    self.test.to_csv(test_path, sep="\t", index=False, header=False)
    self.validation.to_csv(validation_path, sep="\t", index=False,
                           header=False)

    self.combo_df.to_csv(price_path, sep="\t", index=False, header=False)
    self.sess.to_csv(session_path, sep="\t", index=False, header=False)
    self.groups.to_csv(group_path, sep="\t", index=False, header=False)

```

A.2 Shopper

```

def _prepare_data(data: pd.DataFrame):
    """Preprocessing that used to encode all input
    """

    prices = data['price']

    order = data.groupby(['user_id', 'session_id'])['item_id']\

```

```

        .cumcount()

sf = (order.apply(lambda x: 1 / x if x > 0 else 0)
      .to_numpy(dtype='float32'))

obs = (preprocessing.LabelEncoder()
       .fit_transform(data.index)
       .astype('int32'))

items = (preprocessing.LabelEncoder()
         .fit_transform(data['item_id'])
         .astype('int32'))

users = (preprocessing.LabelEncoder()
         .fit_transform(data['user_id'])
         .astype('int32'))

labels = (preprocessing.LabelEncoder()
          .fit_transform(data['item_id'])
          .astype('int32'))

return {'prices': prices,
        'order': order,
        'sf': sf,
        'obs': obs,
        'items': items,
        'users': users,
        'labels': labels}

```

```

class Shopper:
    """Shopper using PyMC3 for generating graphs

```

```

T = trips;
U = users;
C = items;
W = weeks.

"""
def __init__(self,
              data: pd.DataFrame,
              K: int = 50,
              price_dim: int = 10,
              price_dtype: str = 'float32',
              rho_var: float = 1,
              alpha_var: float = 1,
              lambda_var: float = 1,
              theta_var: float = 1,
              delta_var: float = 0.01,
              mu_var: float = 0.01,
              gamma_rate: float = 1000,
              gamma_shape: float = 100,
              beta_rate: float = 1000,
              beta_shape: float = 100):
    """Initializes Shopper instance.

    Args:
        data (Pandas DataFrame):
            Observed trips data (number of trips by 4).
            DataFrame with columns: user_id, item_id, session_id,
            and price.

        K (int):
            Number of latent factors for alpha_c, rho_c, and theta_u;

```


defaults to 50.

`price_dim (int):`

Number of latent factors for price vectors `gamma_u` and `beta_c`;
defaults to 10.

`price_dtype (str):`

The datatype used for prices; defaults to `float32`.

`rho_var (float):`

Prior variance over `rho_c`; defaults to 1.

`alpha_var (float):`

Prior variance over `alpha_c`; defaults to 1.

`theta_var (float):`

Prior variance over `theta_u`; defaults to 1.

`lambda_var (float):`

Prior variance over `lambda_c`; defaults to 1.

`delta_var (float):`

Prior variance over `delta_w`; defaults to 0.01.

`mu_var (float):`

Prior variance over `mu_c`; defaults to 0.01.

`gamma_rate (float):`

Prior rate over `gamma_u`; defaults to 1000.

`gamma_shape (float):`

Prior shape over `gamma_u`; defaults to 100.

```

    beta_rate (float):
        Prior rate over beta_c; defaults to 1000.

    beta_shape (float):
        Prior shape over beta_c; defaults to 100.
"""
# Set data
self.data = data

# Number of items
C = data['item_id'].nunique()
# Number of users
U = data['user_id'].nunique()

# Get preprocessed data variables
data_vars = _prepare_data(data)

logging.info('Building the Shopper model...')
with pm.Model() as shopper:
    # Data
    prices = pm.Data('prices', data_vars['prices'])
    order = pm.Data('order', data_vars['order'])
    sf = pm.Data('sf', data_vars['sf'])
    obs = pm.Data('obs', data_vars['obs'])
    articles = pm.Data('articles', data_vars['articles'])
    users = pm.Data('users', data_vars['users'])
    labels = pm.Data('labels', data_vars['labels'])

    # Latent variables
    # per item interaction coefficients
    rho_c = pm.Normal('rho_c',

```

```

        mu=0,
        sigma=rho_var,
        shape=(K, C),
        dtype='float32')

# per item attributes
alpha_c = pm.Normal('alpha_c',
                    mu=0,
                    sigma=alpha_var,
                    shape=(K, C),
                    dtype='float32')

# per user preferences
theta_u = pm.Normal('theta_u',
                    mu=0,
                    sigma=theta_var,
                    shape=(K, U),
                    dtype='float32')

# per item popularity
lambda_c = pm.Normal('lambda_c',
                    mu=0,
                    sigma=lambda_var,
                    shape=C,
                    dtype='float32')

# per user price sensitivities
gamma_u = pm.Gamma('gamma_u',
                   beta=gamma_rate,
                   alpha=gamma_shape,
                   shape=(price_dim, U),
                   dtype='float32')

# per item price sensitivities
beta_c = pm.Gamma('beta_c',
                  beta=beta_rate,
                  alpha=beta_shape,

```

```

        shape=(price_dim, C),
        dtype='float32')

# Baseline utility per basket per item
# Item popularity + Consumer Preferences - Price Effects
# Note: variation comes from customer index and item prices
psi_tc = pm.Deterministic(
    'psi_tc',
    lambda_c +
    pm.math.dot(theta_u[:, users].T, alpha_c) -
    pm.math.dot(np.log(prices).astype(price_dtype),
        pm.math.dot(gamma_u[:, users].T, beta_c))
)
logging.info('psi_tc shape:
    {}'.format(psi_tc.tag.test_value.shape))

#  $\sum_{j=1}^i [\alpha_{y_{tj}}]$ 
def basket_items_attr(omega_prev, idx, alpha_c, order):
    # If first item in basket
    if tt.eq(order[idx], 0):
        # No price-attributes interaction effects
        omega_ti = tt.zeros(K)
    else:
        omega_ti = omega_prev + alpha_c[:, idx-1]
    return omega_ti

# omega_ti initial value
omega_0 = tt.zeros(K)
omega_ti = tt.vector('omega_ti')
omega_ti, updates = theano.scan(fn=basket_items_attr,
                                outputs_info=omega_0,
                                non_sequences=[obs,

```

```

        alpha_c,
        order],
        n_steps=obs.shape[0])

# Mean utility per basket per item
Psi_tci = pm.Deterministic(
    'Psi_tci',
    psi_tc + pm.math.dot(
        sf[obs],
        pm.math.dot(omega_ti[obs-1, :], rho_c))
)
logging.info('Psi_tci shape: {}'.format(
    Psi_tci.tag.test_value.shape))
# Softmax likelihood p(y_ti = c | y_t0, y_t1, ..., y_ti-1)
p = pm.Deterministic(
    'p',
    tt.nnet.softmax(Psi_tci[articles])
)
logging.info('p shape: {}'.format(p.tag.test_value.shape))
y = pm.Categorical('y', p=p, observed=labels)
logging.info('y shape: {}'.format(y.tag.test_value.shape))

logging.info("Done building the Shopper model.")
# Set shopper to model attribute
self.model = shopper

def fit(self,
        N,
        diff='relative',
        random_seed=42,
        **kwargs):
    """Estimate parameters using Bayesian inference. using ADVI
    Returns ShopperResults instance.

```

Args:

`N (int):`

Number of iterations (ADVI).

`diff (str):`

Requires method to be ADVI. The difference type used to check convergence in the mean of the ADVI approximation

`random_seed (int):`

Random seed; defaults to 42.

"""

`model = self.model`

`with model:`

`callback = CheckParametersConvergence(diff=diff)`

`res = pm.fit(n=N,
 method='advi',
 callbacks=[callback],
 random_seed=random_seed,
 **kwargs)`

`return ShopperResults(model, res)`

`class ShopperResults:`

"""Results class for a fitted Shopper model.

Attributes:

`model (PyMC3 Model):`

Shopper model.

`res (PyMC3 results instance):`

If MCMC, then requires `arviz.InferenceData` or

```

        MultiTrace.InferenceData. Else if ADVI, then
        requires pymc3.variational.opvi.Approximation.
"""
def __init__(self, model, res):
    self.model = model
    self.res = res

def summary(self, **kwargs):
    """Returns text-based output of common posterior statistics.

    Requires 'draws' (sample size to be drawn from posterior
        distribution)
    to be set in kwargs if model was fitted with ADVI.
    """
    res = self.res
    if 'variational' in str(type(res)):
        logging.info('Sampling from posterior distribution...')
        trace = res.sample(draws=kwargs['draws'])
        logging.info('Sampling complete.')
        logging.info('Computing posterior statistics...')
        summary = az.summary(trace, kind='stats')
    else:
        summary = az.summary(res)
    return summary

def trace_plot(self, **kwargs):
    """Returns the trace plot.

    Requires 'draws' (sample size to be drawn from posterior
        distribution)
    to be set in kwargs if model was fitted with ADVI.
    """

```

```

res = self.res
if 'variational' in str(type(res)):
    logging.info('Sampling from posterior distribution...')
    trace = res.sample(draws=kwargs['draws'])
    logging.info('Sampling complete.')
    plot = az.plot_trace(trace)
else:
    plot = az.plot_trace(res)
return plot

def rhat(self):
    """Returns the Gelman-Rubin statistic.

    Requires the Shopper model to be fitted with
    MCMC sampling.
    """
    return az.summary(self.res)

def energy_plot(self):
    """Returns energy plot to check for convergence.

    Commonly used for high-dimensional models where it
    is too cumbersome to examine all parameter's traces.

    Requires the Shopper model to be fitted with
    MCMC sampling.
    """
    return az.plot_energy(self.res)

def elbo_plot(self):
    """Returns trace plot of ADVI's objective function (ELBO).

    Requires the Shopper model to be fitted with ADVI.

```



```

"""

fig = plt.figure()
plt.plot(self.res.hist)
plt.ylabel('ELBO')
plt.xlabel('n iterations')
return fig

def predict(self, data, random_seed=42, **kwargs):
    """Returns predicted probabilities of outcomes for samples in X.
    """
    model = copy.deepcopy(self.model)
    res = self.res
    data_vars = _prepare_data(data)
    data_vars.pop('labels') # Remove labels
    with model:
        # Pass new values to model
        pm.set_data(data_vars)
        # Use the updated values and
        # predict outcomes and probabilities
        if 'variational' in str(type(res)):
            logging.info('Sampling from posterior distribution...')
            trace = res.sample(draws=kwargs['draws'])
            logging.info('Sampling complete.')
            posterior_predictive = pm.sample_posterior_predictive(
                trace,
                random_seed=random_seed
            )
        else:
            posterior_predictive = pm.sample_posterior_predictive(
                res,
                random_seed=random_seed
            )

```

```

        return posterior_predictive

def score(self, data):
    """Returns the mean accuracy on the given test data and labels.
    """
    pass

if __name__ == "__main__":
    pass

```

A.3 Baseline

```

def _prepare_data(df, week):
    trans_rd = df[(df["week"] > week) & (df["week"] <= week + WEEK_MAX)]
    trans_rd = trans_rd.groupby("customer_id").agg({"article_id": list,
                                                    "week": list}).reset_index()
    trans_rd.rename(columns={"week": 'week_h'}, inplace=True)

    target_df = df[df["week"] == week]
    target_df = target_df.groupby("customer_id").agg({"article_id":
                                                       list}).reset_index()
    target_df.rename(columns={"article_id": "target"}, inplace=True)
    target_df["week"] = week

    return target_df.merge(trans_rd, on="customer_id", how="left")

```

```

class Baseline(Dataset):
    def __init__(self, df, seq_len, is_test=False):
        self.df = df.reset_index(drop=True)
        self.seq_len = seq_len

```

```

self.is_test = is_test

def __len__(self):
    return self.df.shape[0]

def __getitem__(self, index):
    row = self.df.iloc[index]

    if self.is_test:
        target = torch.zeros(2).float()
    else:
        target = torch.zeros(len(article_ids)).float()
        for t in row.target:
            target[t] = 1.0

    article_hist = torch.zeros(self.seq_len).long()
    week_hist = torch.ones(self.seq_len).float()

    if isinstance(row.article_id, list):
        if len(row.article_id) >= self.seq_len:
            article_hist =
                torch.LongTensor(row.article_id[-self.seq_len:])
            week_hist =
                (torch.LongTensor(row.week_history[-self.seq_len:]) -
                 row.week) / WEEK_MAX / 2
        else:
            article_hist[-len(row.article_id):] =
                torch.LongTensor(row.article_id)
            week_hist[-len(row.article_id):] =
                (torch.LongTensor(row.week_history) - row.week) /
                WEEK_MAX / 2

```

```
return article_hist, week_hist, target
```

A.4 Image Feature Vector

```
def load_images_from_folder(folder, image_size = (1166, 1750)):
    images = []
    for filename in os.listdir(folder):
        #         img = cv2.imread(os.path.join(folder,filename))
        img = load_img(os.path.join(folder, filename),
                        target_size=image_size)
        img = img_to_array(img)
        img = img.reshape((1,) + img.shape)
        if img is not None:
            images.append(img)
    return images

def get_all_images(w, h):
    images1 = load_images_from_folder('./hm/images/010', (w,h))
    images2 = load_images_from_folder('./hm/images/013', (w,h))
    images3 = load_images_from_folder('./hm/images/015', (w,h))
    images4 = load_images_from_folder('./hm/images/018', (w,h))
    images5 = load_images_from_folder('./hm/images/020', (w,h))
    images6 = load_images_from_folder('./hm/images/027', (w,h))
    all_imgs_arr = np.array([images1 + images2 + images3 + images4 + images5
                             + images6])

    # images1 = load_images_from_folder('./hm/images/010', (w, h))
    #
    # all_imgs_arr = np.array([images1])
```

```

    # print(all_imgs_arr.shape)
    return all_imgs_arr

def create_model(model_file):
    # loading vgg16 model and using all the layers until the 2 to the last
    # to use all the learned cnn layers

    ssl._create_default_https_context = ssl._create_unverified_context
    vgg = VGG19(include_top=True)
    model2 = Model(vgg.input, vgg.layers[-2].output)
    model2.save(model_file) # saving the model just in case
    return model2

def get_model(model_file):
    model = None
    try:
        model = load_model(model_file)
    except:
        model = create_model(model_file)
    return model

def get_preds(all_imgs_arr, model):
    preds_all = np.zeros((len(all_imgs_arr), 4096))
    for j in range(all_imgs_arr.shape[0]):
        preds_all[j] = model.predict(all_imgs_arr[j])

    return preds_all

def load_images_preds(numpy_filepath):
    # This file is a saved file that has all the trained images and their
    # corresponding prediction from vgg16 model

```

```

data = np.load(numpy_filepath)
img = data['images']
preds = data['preds']
return img, preds

def show_img(array):
    array = array.reshape(224,224,3)
    numpy_image = img_to_array(array)
    plt.imshow(np.uint8(numpy_image))
    plt.show()

def load_images_from_file(filepath):

    img = load_img(filepath, target_size=(224, 224))
    img = img_to_array(img)
    img = img.reshape((1,) + img.shape)
    return img

def get_nearest_neighbor_and_similarity(preds1, K):
    dims = 4096
    n_nearest_neighbors = K+1
    trees = 10000
    file_index_to_file_vector = {}

    # build ann index
    t = AnnoyIndex(dims, metric='angular')
    for i in range(preds1.shape[0]):

        file_vector = preds1[i]
        file_index_to_file_vector[i] = file_vector
        t.add_item(i, file_vector)
    t.build(trees)

```

```

for i in range(preds1.shape[0]):
    master_vector = file_index_to_file_vector[i]

    named_nearest_neighbors = []
    similarities = []
    nearest_neighbors = t.get_nns_by_item(i, n_nearest_neighbors)
    for j in nearest_neighbors:
        # print (j)
        neighbor_vector = preds1[j]
        similarity = 1 - spatial.distance.cosine(master_vector,
            neighbor_vector)
        rounded_similarity = int((similarity * 10000)) / 10000.0
        similarities.append(rounded_similarity)
    return similarities, nearest_neighbors

def get_similar_images(similarities, nearest_neighbors, images1):
    j = 0
    for i in nearest_neighbors:
        show_img(images1[i])
        print (similarities[j])
        j+=1

```

Appendix B

Figures

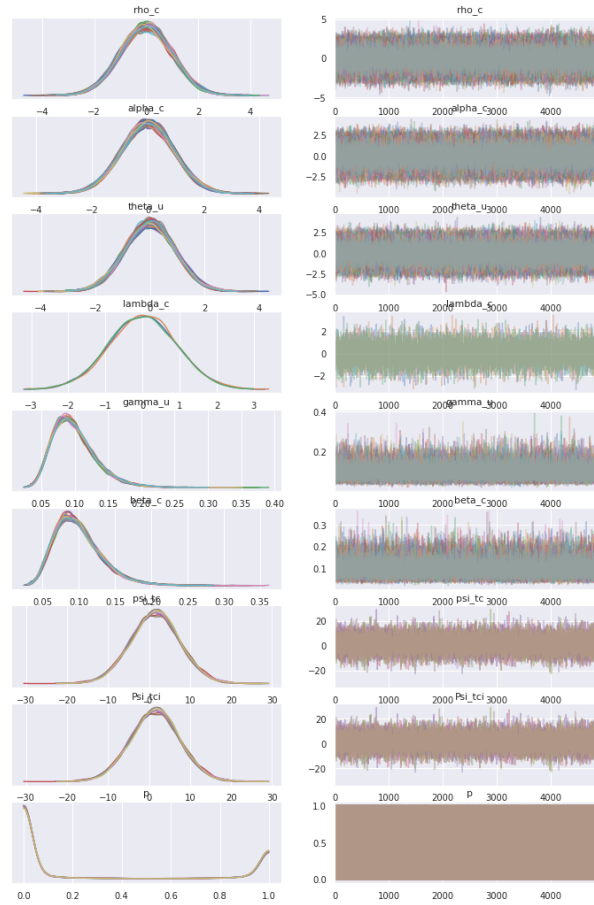


Figure B-1: PDF Graph

Fig. B-1 is the graph of probability density function for the variables.

Bibliography

- [1] H&M personalized fashion recommendations. <https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations>. Accessed: 2022-05-09.
- [2] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB '94*, page 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.
- [3] Dr. Shahid Bhat, Keshav Kansana, and Jenifur Majid. A review paper on e-commerce. 02 2016.
- [4] Rsquared Academy Blog. Practical introduction to market basket analysis – association rules: R-bloggers, May 2019.
- [5] Thomas G. Dietterich. *Machine Learning*, page 1056–1059. John Wiley and Sons Ltd., GBR, 2003.
- [6] Alan Eckhardt. Various aspects of user preference learning and recommender systems. volume 471, pages 56–67, 01 2009.
- [7] V. K. Gancheva. Market basket analysis of beauty products. 2013.
- [8] Prem Gopalan, Jake M. Hofman, and David M. Blei. Scalable recommendation with poisson factorization, 2013.
- [9] Veronika Simoncikova Haseeb Warsi, Benedict Becker.
- [10] Jochen Hipp, Ulrich Güntzer, and Gholamreza Nakhaeizadeh. Algorithms for association rule mining — a general survey and comparison. *SIGKDD Explor. Newsl.*, 2(1):58–64, jun 2000.
- [11] Manpreet Kaur and Shivani Kang. Market basket analysis: Identify the changing trends of market data using association rule mining. *Procedia Computer Science*, 85:78–85, 2016. International Conference on Computational Modelling and Security (CMS 2016).
- [12] Puneet Manchanda, Asim Ansari, and Sunil Gupta. The "shopping basket": A model for multicategory purchase incidence decisions. *Marketing Science*, 18(2):95–114, 1999.

- [13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space, 2013.
- [14] Rafid Mostafiz, Mohammad Motiur Rahman, A Islam, and Saeid Belkasim. machine learning knowledge extraction focal liver lesion detection in ultrasound image using deep feature fusions and super resolution. *Machine Learning and Knowledge Extraction*, 2, 07 2020.
- [15] Maja R. Rudolph, Francisco J. R. Ruiz, Stephan Mandt, and David M. Blei. Exponential family embeddings, 2016.
- [16] Francisco J. R. Ruiz, Susan Athey, and David M. Blei. Shopper: A probabilistic model of consumer choice with substitutes and complements, 2017.
- [17] Kunal Shah, Akshaykumar Salunke, Saurabh Dongare, and Kisandas Antala. Recommender systems: An overview of different approaches to recommendations. In *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–4, 2017.
- [18] Osvaldo Simeone. A very brief introduction to machine learning with applications to communication systems. *IEEE Transactions on Cognitive Communications and Networking*, 4(4):648–664, 2018.
- [19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.
- [20] M.J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.
- [21] Mohammed J Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. Technical report, USA, 1997.