CS584-Machine Learning: Project Final Report Product Recommendation Model

Xiaoting Zhou
Illinois Institute of Technology
Chicago, Illinois, USA
xzhou70@hawk.iit.edu
A20503290

Nevil Jack Denis
Illinois Institute of Technology
Chicago, Illinois, USA
psubramanian@hawk.iit.edu
A20474215

Hongbo Wang
Illinois Institute of Technology
Chicago, Illinois, USA
hwang218@hawk.iit.edu
A20524770

ABSTRACT

The Product Recommendation Model (PRM) [1] is a powerful tool for identifying associations between frequently co-occurring entities and objects, such as items in a shopper's basket. This project aims to help Instacart utilize its transaction data to understand customer purchase patterns, identify frequently purchased items, and maintain adequate product stock. We aim to identify clusters and subgroups of customers with similar purchase behavior and provide actionable recommendations for improving revenue and customer experience through segmentation and prediction models. The Product Recommendation Model (PRM) is used to analyze retail basket or transaction data and identify strong rules based on measures of interestingness. Our analysis will enable Instacart to enhance the user experience by suggesting the next likely product to purchase during the order process. Additionally, we will outline a marketing strategy for personalized communications to customers highlighting predicted products. This project provides valuable insights into PRM and its potential applications for improving business operations in the retail industry.

CCS CONCEPTS

• **Computing methodologies** → **Inductive logic learning**.

KEYWORDS

items, association, analyze, store, customer, transaction, data, predict

ACM Reference Format:
Xiaoting Zhou, Nevil Jack Denis, and Hongbo Wang. 2018. CS584-Machine Learning: Project Final Report Product Recommendation Model. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation emai (Conference acronym 'XX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 OVERVIEW

This project aims to perform a market basket analysis of Instacart's transaction data to understand customer purchase patterns and provide actionable recommendations for improving revenue and customer experience. By utilizing the Product Recommendation Model (PRM), we will identify frequently purchased items and clusters of customers with similar purchase behavior. Our analysis will enable Instacart to suggest the next likely product to purchase during the order process, enhancing the user experience. Additionally, we will provide insights into PRM and its potential applications for improving business operations in the retail industry. The outcomes of this project can inform decision-making regarding product placement, inventory management, and personalized communications with customers highlighting predicted products (Figure 1).

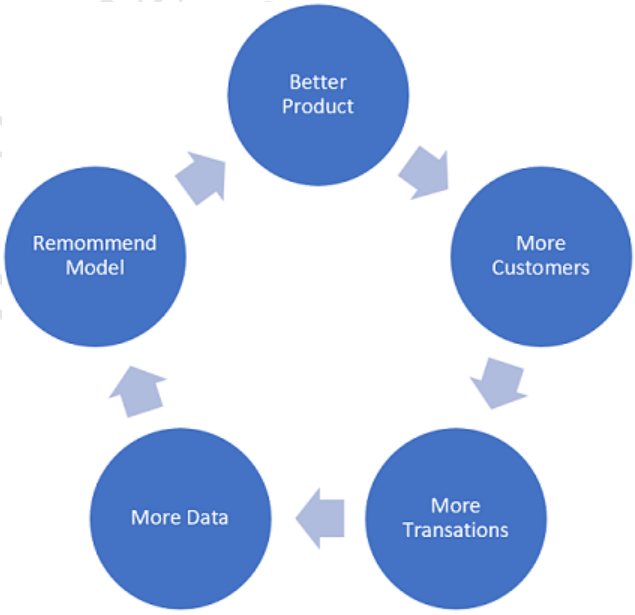

Figure 1: Overview of Product Recommendation Model

2 PROBLEM STATEMENT

Instacart is an American company that provides online grocery delivery and pick-up services through a website and mobile app. In order to improve the company's sales, we aim to develop a model that predicts which items will be the best seller in the future target period. The need to build a prediction model for Instacart arises from the desire to understand customer purchase behavior and improve business operations. By analyzing transaction data and identifying frequently purchased items, Instacart can optimize product placement, shelf arrangements, and up-sell, cross-sell, and bundling opportunities, ultimately improving sales and customer satisfaction.

Conference acronym 'XX, June 03–05, 2018, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

The prediction model can further enhance the user experience by suggesting the next likely product to purchase during the order process. This personalized recommendation system can increase the chances of customers purchasing additional items, leading to higher order values and increased revenue for Instacart. Moreover, by identifying clusters and subgroups of customers with similar purchase behavior, Instacart can segment its customer base and provide targeted communications, highlighting predicted products and improving customer loyalty.

3 RELATED WORK

Recommender systems typically involve a two-stage process: retrieval and ranking. The retrieval model selects a preliminary set of candidates, while the ranking model identifies the best possible recommendations. Recent research has focused on improving the accuracy and efficiency of recommender systems, such as graph-based and deep learning-based approaches. However, there is still a gap in the literature on how to handle cold-start and data sparsity problems. These problems can arise when there is limited user data available or when a new item is introduced, making it difficult for traditional techniques like matrix factorization to generate accurate recommendations. Addressing these issues is critical for building more effective recommender systems that can improve user satisfaction and engagement.

4 DATASET AND FEATURES

4.1 Data Preparation

The data utilized in this project were gathered from Kaggle, specifically the Simplified Instacart Market Basket Dataset. This dataset consists of a set of interrelated files that document the orders made by customers over a period of time. The data set comprises an anonymous sample of over 3 million grocery orders placed by more than 200,000 Instacart users. Each user's order details, including the items purchased, the order's date and week, and the time elapsed between orders, are included in the dataset. The number of order details for each user ranges from 4 to 100 (Figure 2).

The description of the data and its columns/features in the dataset is mentioned below. Each entity (customer, product, order, aisle, etc.) has an associated unique id and its related data.

- `Aisles`: This file contains different aisles and there are a total of 134 unique aisles.
- `Departments`: This file contains different departments and there are a total of 21 unique departments.
- `Orders`: This file contains all the orders made by different users.
- `Products`: This file contains the list of a total of 49688 products and their aisle as well as the department.
- `Prior`: This file gives information about which products were ordered and in which order they were added to the basket. It also tells us that if the product was reordered or not.
- `Prior`: The structure is as same as the Prior data set, and these two data sets combine together as the whole order information (data set).

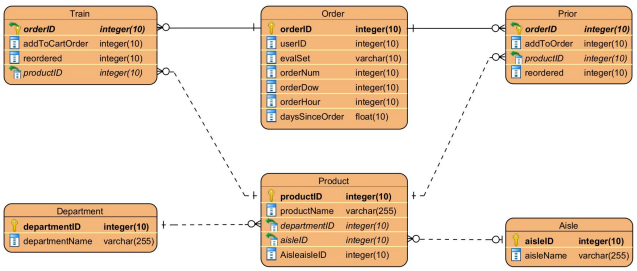

Figure 2: Dataset Diagram

4.2 Data Processing

There are 4 to 100 orders with product purchase sequences for each user. The objective is to forecast the items that will be the best seller and find the top 100 popular items.

We conducted various exploratory data analyses to gain a better understanding of the data, as feature selection plays a crucial role in determining model performance. To enhance the data processing, we implemented the following steps.

- `Step 1`: Read the CSV files of the data set.
- `Step 2`: Data analysis and select features of data sets.
- `Step 3`: Print out the selected features and merge them into a CSV file.

Step 1: Read the CSV files of the data sets (Figure 3).

```
# Read the CSV file
prior_data = pd.read_csv("./database/order_products__prior.csv")
train_data = pd.read_csv("./database/order_products__train.csv")
orders_data = pd.read_csv("./database/orders.csv")
products_data = pd.read_csv("./database/products.csv")

print(orders_data.head())

prior_data = prior_data.values
train_data = train_data.values
orders_data = orders_data.values
products_data = products_data.values
```

```
   order_id  user_id eval_set  order_number  order_dow  order_hour_of_day  \
0   2539329        1    prior             1          2                  8
1   2398795        1    prior             2          3                  7
2    473747        1    prior             3          3                 12
3   2254736        1    prior             4          4                  7
4    431534        1    prior             5          4                 15

   days_since_prior_order
0                     NaN
1                    15.0
2                    21.0
3                    29.0
4                    28.0
```

Figure 3: The screenshot of processing code fragment

Step 2: Data analysis and select features of data sets.

We thoroughly cleaned and improved the dataset to showcase its various aspects. Additionally, we delved deeper into the data to identify the product offerings within the Instacart data, enabling us to select highly relevant features (Figure 4).

Initially, we investigated the frequently reordered items. We display the product's name and the quantity purchased during this past order. The results illustrate that the top two best-selling items are bananas, with most of these products being organic.

We will begin by analyzing the products ordered and grouping them by hour and day (as shown in Figures 4 and 5). Our aim is to compare the number of products ordered by Instacart customers per hour and per day. The analysis indicates that during the first 5-6 hours of the day, the number of products ordered is significantly lower. However, between 7 and 20 hours, there is a steady increase in the number of products ordered, which then decreases after 20

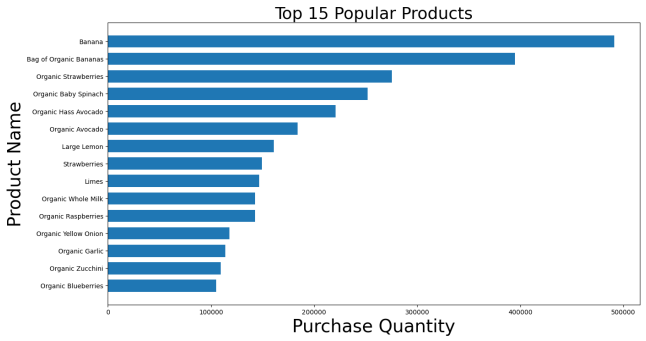

Figure 4: Top 15 popular products

hours. This suggests that the middle of the day is the busiest time for ordering products, while the opening and closing times see less activity.

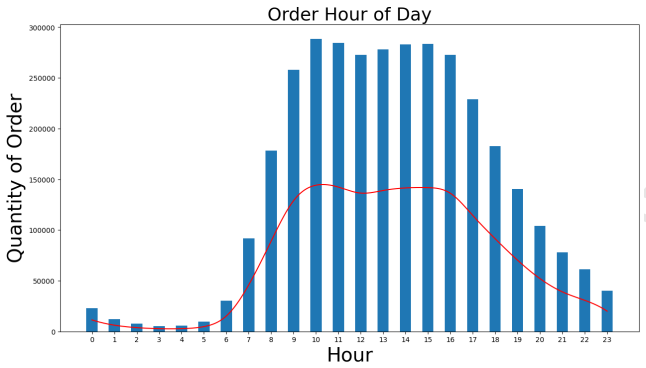

Figure 5: Order Hour of Day

The graph below illustrates the number of products ordered each week across 7 days. The results clearly show that at the start of the week, there are more product orders, which then decrease towards the middle of the week before stabilizing towards the end of the week.

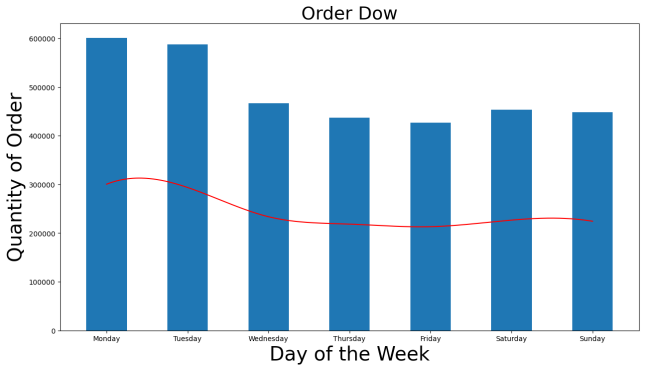

Figure 6: Order Dow

Step 3: Print out the selected features and merge them into a CSV file.

Popular reordered product data set.

```
print("order_id product_id  reordered")
for i in range(0,10):
  print(dataset[i])

print(len(orders_data))
```

```
order_id        product_id      reordered
[0 0 0]
[     1 49683     0]
[     1 13176     0]
[     1 47209     0]
[     1 22035     1]
[     2 28985     1]
[     2 17794     1]
[     3 24838     1]
[     3 21903     1]
[     3 46667     1]
3421083
```

Figure 7: Code fragment of popular reordered product

The order information per hour of one day.

```
print("order_id  user_id eval_set  order_number  order_dow  order_hour_of_day days_since_prior_order")
for i in range(0,10):
  print(orders_data[i])

print(len(orders_data))
```

```
order_id  user_id eval_set  order_number  order_dow  order_hour_of_day days_since_prior_order
[1 112108 'train' 4 4 10 9.0]
[2 202279 'prior' 3 5 9 8.0]
[3 205970 'prior' 16 5 17 12.0]
[4 178520 'prior' 36 1 9 7.0]
[5 156122 'prior' 42 6 16 9.0]
[6 22352 'prior' 4 1 12 30.0]
[7 142903 'prior' 11 2 14 30.0]
[8 3107 'prior' 5 4 6 17.0]
[9 139016 'prior' 14 0 19 5.0]
[10 135442 'prior' 4 6 8 8.0]
3421083
```

Figure 8: Order Dow

The order information per day for one week.

```
print("order_id  product_id  order_dow  order_hour_of_day  reordered")
for i in range(0,15):
  print(final_dataset[i])
```

```
order_id  product_id  order_dow  order_hour_of_day  reordered
[0 0 0 0 0]
[     1 49683     4    10     0]
[     1 13176     4    10     0]
[     1 47209     4    10     0]
[     1 22035     4    10     1]
[     2 28985     5     9     1]
[     2 17794     5     9     1]
[     3 24838     5    17     1]
[     3 21903     5    17     1]
[     3 46667     5    17     1]
[     5 13176     6    16     1]
[     5 27966     6    16     1]
[     5 23909     6    16     1]
[     5 47209     6    16     0]
[     9 31506     0    19     1]
```

Figure 9: Code fragment of order information of one week

Merger the related features into one data set.

```
def new_product_table():
  dic = {}
  new_products = []
  j = 0
  for i in range(0, len(products_data)):
    if products_data[i][0] == top_100_popular_productsID[j]:
      new_products.append([top_100_popular_productsID[j],products_data[i][1]])
      j = j + 1
    if j == 100:
      break
  return new_products

new_products = new_product_table()
```

Figure 10: Order Dow

Save the new dataset to a new CSV file.
2 new CSV files are created after the data processing.

Save the final version as dataset.

order_id product_id order_dow order_hour_of_day reordered

```
np.savetxt('./database/dataset.csv', final_dataset, delimiter=',',fmt='%d')
```

```
name = ['productID','productName']
new_products_table = pd.DataFrame(columns=name,data=new_products)
new_products_table.to_csv('./database/new_porducts.csv')
```

Figure 11: Save as CSV file

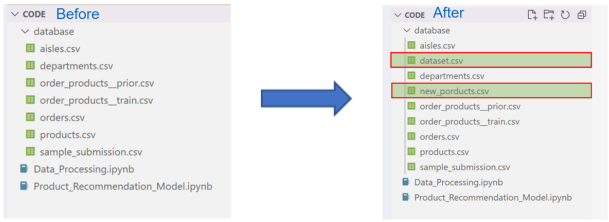

Figure 12: Two new CSV files

5 IMPLEMENTATION

In this part, it discusses how to implement data set segmentation, mini-batch generation method, product recommendation model, training process, validation process, and testing process. Through these processes, the performance of the final model is shown in (Figure 13), and the test set accuracy is 10.725% in 100 class classification.

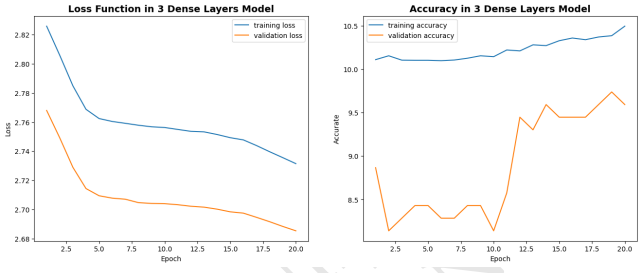

Figure 13: The loss and accuracy of train and validation sets in 20 epoch

5.1 Splitting of training, validation, and testing

The split_data function serves to partition the entire data set into training, validation, and test sets, allocating 98%, 1%, and 1% of the data, respectively. The function begins by generating a set of lists using the numpy built-in function np.random.permutation, which produces a sequence of non-repeating integers that spans the length of the data set. These integers are sorted based on random seeds, which enables the random composition of the three data sets (train, validation, and test) for each run of the function.

After generating the set of lists, the split_data function proceeds to allocate memory space based on the size of each data set (i.e., train, valid, and test). Subsequently, the function utilizes a for loop to traverse the list of randomly generated numbers, employing the values in the list as indices to access and assign elements from the original data set to the respective train, valid, and test sets.

Split the data into training, validation and testing set.

```
def split_data():
    """
    This is split function.
    Split X into X_train, X_valid, X_test
    Split y into y_train, y_valid, y_test
    The number of X_train : X_valid : X_test is [98 : 1 : 1]
    len(X_train) = 67476
    len(X_valid) = 688
    len(X_test) = 688
    """
    random = np.random.permutation(len(X))    # random is a list that store the random number
    train_size = int(0.98 * len(X))
    valid_size = int(0.01 * len(X))
    test_size = int(0.01 * len(X))

    X_train = torch.zeros(size=(train_size,102,1), dtype=torch.float32)
    X_valid = torch.zeros(size=(valid_size,102,1), dtype=torch.float32)
    X_test = torch.zeros(size=(test_size+2,102,1), dtype=torch.float32)
    y_train = torch.zeros(train_size)
    y_valid = torch.zeros(valid_size)
    y_test = torch.zeros(test_size+2)

    for i in range(0, len(random)):
        index = random[i]
        if(i < train_size):
            X_train[i] = X[index]
            y_train[i] = y[index]
        elif(i >= train_size and i < train_size + valid_size):
            X_valid[i-train_size] = X[index]
            y_valid[i-train_size] = y[index]
        else:
            X_test[i-train_size-valid_size] = X[index]
            y_test[i-train_size-valid_size] = y[index]

    return X_train, y_train, X_valid, y_valid, X_test, y_test

X_train, y_train, X_valid, y_valid, X_test, y_test = split_data()
```

Figure 14: The implementation of splitting train, valid, test data sets

5.2 Splitting of the training set into mini-batch set

As the size of the training dataset is typically quite large, loading it into the model in its entirety with a batch size equal to the dataset length can be inefficient, while a batch size of 1 can be excessively time-consuming due to the need for frequent backpropagation. To overcome this issue, the batch function utilizes a hyperparameter, batch size, with a default value of 32, to split the training dataset into a series of smaller subsets, known as mini-batches. By employing the modulus operation, the resulting modulo values serve as indices for selecting and grouping the training data into mini-batches, which are subsequently appended to a list, enabling efficient batch processing of the entire dataset.

5.3 The Product Recommendation Model

The model is based on a three-layer fully connected network. The first layer, also known as the input layer (the 0 layer), consists of 102 input units. The first 100 input units correspond to the purchased items, with a binary value assigned to each unit such that a value of 1 represents the purchase of the corresponding item, while all other units are assigned a value of 0.

```
class ProductRecommendModel(torch.nn.Module):
    def __init__(self):
        super(ProductRecommendModel,self).__init__()
        self.layer1 = torch.nn.Linear(102,102)
        self.layer2 = torch.nn.Linear(102,101)
        self.layer3 = torch.nn.Linear(101,100)

    def forward(self,x):
        x = x.view(-1,102)
        x = F.relu(self.layer1(x))
        x = F.relu(self.layer2(x))

        out = self.layer3(x)

        return out


product_recommend_model = ProductRecommendModel().to(device)
criterion = torch.nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(product_recommend_model.parameters(),lr=0.01)
```

Figure 15: The implementation of model code fragment

Additionally, the 101st input unit corresponds to the week of purchase, represented as an integer value, while the 102nd input unit represents the time of purchase, recorded in a 24-hour clock format. The activation function is ReLu.

The middle layer of the model contains 102 input units and 101 output units. The activation function used in this layer is Rectified Linear Unit (ReLU). ReLU is a non-linear activation function that is widely used in neural networks due to its ability to address the vanishing gradient problem and its computational efficiency. The ReLU function returns a value of 0 for negative inputs and the input value itself for positive inputs, thus allowing the model to learn complex, non-linear relationships between the input and output units in the middle layer.

The output layer of the model contains 101 input units and 100 output units.

The model employs the Cross-Entropy Loss function, implemented through the PyTorch library's torch.nn.CrossEntropyLoss module, as its chosen loss function. This loss function is commonly used for multi-class classification tasks, such as the one at hand, and measures the dissimilarity between the predicted class probabilities and the actual class labels in the dataset. Additionally, the model uses the Stochastic Gradient Descent (SGD) optimizer, implemented through the torch.optim.SGD module in PyTorch, to update the model's parameters during training. SGD is a widely used optimization algorithm that iteratively updates the model's parameters in the direction of the negative gradient of the loss function, with the goal of minimizing the loss and improving the model's accuracy over time.

5.4 Training Process

During the training phase, the model iteratively processes each minibatch of data in a single epoch. At the start of each epoch, the optimizer is initialized and the minibatch is fed into the model as input. The resulting output is then compared with the actual values, and the difference is measured as the loss. The backpropagation algorithm is then applied to update the model's parameters and optimize its weights, thereby reducing the loss and improving its accuracy. This process is repeated for all minibatches in the epoch, until the model has been trained on the entire training dataset. Ultimately, this training function enables the model to learn and generalize from the input data, improving its performance on the validation and test datasets.

```python
def train(epoch):
  total = len(X_train)

  with tqdm(range(0, len(X_train_batch))) as loop:
    for i in loop:
      loop.set_description(f"Epoch {epoch}")
      optimizer.zero_grad()
      outputs = product_recommend_model(X_train_batch[i].to(device))
      loss = criterion(outputs, y_train_batch[i].to(device))
      loss.backward()
      optimizer.step()

  correct = 0
  total = len(X_train)
  loss = 0.0
  with torch.no_grad():
    outputs = product_recommend_model(X_train)
    loss = criterion(outputs, y_train)
    _, predicted = torch.max(outputs.data, dim=1)
    for i in range(0, len(predicted)):
      if(predicted[i] == y_train[i]):
        correct += 1
  train_loss.append(loss.cpu().numpy())
  train_accuracy.append(100*correct/total)

```

Figure 16: The implementation of training process

After the initial training process is completed, the model's performance is evaluated by repeating the same process without applying gradient descent. The resulting loss and accuracy are then recorded and appended to a list, providing insights into the model's generalization capabilities and overall effectiveness.

5.5 Validation Process

The process for evaluating the model's performance without gradient descent is similar to that of the corresponding code fragment within the train function.

```python
def valid(epoch):
  correct = 0
  total = len(X_valid)
  loss = 0.0
  with torch.no_grad():
    outputs = product_recommend_model(X_valid)
    loss = criterion(outputs, y_valid)
    _, predicted = torch.max(outputs.data, dim=1)
    for i in range(0, len(predicted)):
      if(predicted[i] == y_valid[i]):
        correct += 1

  valid_loss.append(loss.cpu().numpy())
  valid_accuracy.append(100*correct/total)

```

Figure 17: The implementation of validation process

5.6 Testing Process

After training for all epochs, load the test set into the model and get the accuracy.

```python
def test():
  correct = 0
  total = len(X_test)
  with torch.no_grad():
    outputs = product_recommend_model(X_test)
    _, predicted = torch.max(outputs.data, dim=1)
    for i in range(0, len(predicted)):
      if(predicted[i] == y_test[i]):
        correct += 1
  print(f'Accuracy on test set: {100*correct/total:.3f}%')
```

Figure 18: The implementation of testing process

6 RESULT

6.1 Model Evaluation and Validation

We opted for Root Mean Square Error (RMSE) as the chosen evaluation metric. Python libraries like NumPy, pandas, matplotlib, and torch have been utilized in this paper's implementation as they operate on the data set and generate relevant outcomes.

The accuracy of the model is impacted by the number of epochs and the dense layer. Through several attempts, it has been observed that the loss function diminishes as the number of epochs increases.

We observed that the model's accuracy increased during the initial epochs, but plateaued after a certain point. We attempted to improve the model's performance by adjusting the learning rate, changing the number of nodes in the hidden layers, and modifying the model architecture. However, these changes did not lead to a significant improvement in the model's accuracy.

The final accuracy on the test data set is 10.725% in 100-class classification.

6.2 Justification

Our model's suboptimal performance could be attributed to several factors, including the limited size of the dataset, the complexity

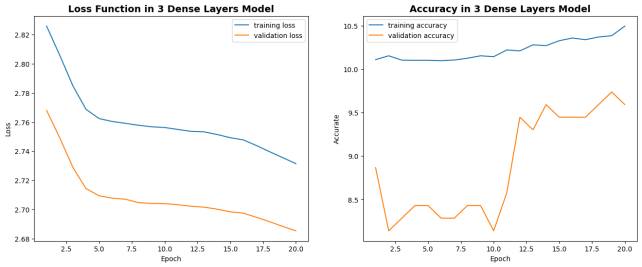

Figure 19: The loss and accuracy of train and validation sets in 20 epoch

of the task, and the hyperparameters of the model. Additionally, the presence of class imbalance in the data may have affected the model's ability to accurately predict certain classes. Further analysis is needed to better understand these factors and improve the model's performance.

7 CONCLUSION AND FUTURE WORK

In this model, we use a 3-layer fully-connected network as the model and achieve an accuracy of 10.725% on 100-class classification. At the same time, we realize that the accuracy rate is not very good, so we have two directions in future work to enhance the accuracy rate of prediction. One is to add more intermediate layers and add more features. The other is to replace it with a better model to obtain higher prediction accuracy.

Overall, improving model accuracy requires careful experimentation with different architectures, hyper-parameters, and pre-processing techniques, an ongoing process that requires constant refinement and improvement.

ACKNOWLEDGMENTS

We are indebted to Professor Yan Yan, who taught CS487, a course on Machine Learning at the Department of Computer Science at the Illinois Institute of Technology. Under his guidance, we gained a comprehensive understanding of machine learning algorithms and the implementation of their underlying principles. Professor Yan's expertise and instruction were invaluable in deepening our knowledge and skills in this field.

Same, we would like to express our gratitude to Changchang Sun and Bin Xie for their invaluable support as teaching assistants. Their prompt and effective response to our queries has been crucial in resolving our academic concerns

REFERENCES

A WEBSITE LINKS

- https://www.sciencedirect.com/topics/computer-science/market-basket-analysis
- https://www.educba.com/association-rules-in-data-mining/
- https://www.datapine.com/blog/data-analysis-methods-and-techniques/
- https://towardsdatascience.com/beginners-guide-to-xgboost-for-classification-problems-50f75aac5390

B README FILE

Name: CS584_Machine-Learning_Project

Team Member:

- Hongbo Wang
- Xiaoting Zhou
- Nevil Jack Denis

Environment:

- Anaconda 23.3.1
- Python 3.11.0

Details:

- Type: term project
- Professor: Yan Yan
- Developing Language: Python
- Project Name: Product Recommendation Model
- Time: April/28/2023
- Description: Through a 30% fully connected neural network test as a product recommendation model, the accuracy of the model is tested by splitting into the train, validation, and test data set, and among 100 types of products, the accuracy of predicting the next purchased product reaches 10.725%.

Dependencies:

- Kaggle data set: data set

Install package:

- `troch`: 1.13.1
- `pandas`: 1.5.3
- `numpy`: 1.23.5
- `tdmq`: 3.7.1
- `Documentation`: 4.65.0

Run Program:

- Download Anaconda and necessary environment
- Download the data set
- Run Data_Processing.ipynb file (First)
- Run Product_Recommendation_Model.ipynb file (second)

(First): The model needs data processing first

Example:

. 1. The screenshot of the model.

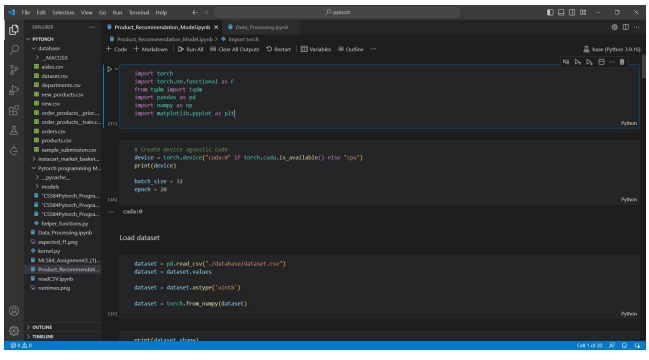

Figure 20: The model of PRM

2. The screenshot of model train and validation loss, and train and validation accuracy.

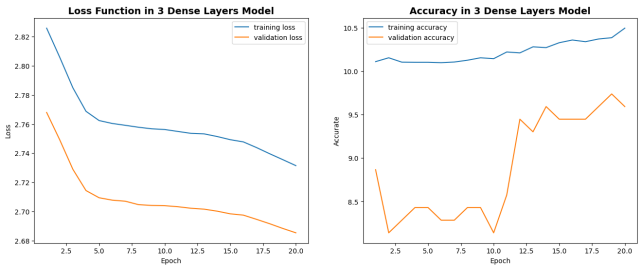

Figure 21: The screenshot of model performance in each epoch

C SOURCE CODE

CS584_Machine-Learning_Project
https://github.com/May-Xiaoting-Zhou/CS584_Machine-Learning_Project

Received 18 January 2023; revised 27 March 2023; accepted 31 March 2023