CPTS315 Final Project

Product recommendation system using frequent itemsets and Gradient boosting

Alex Hunt

May 3, 2023

1 Introduction

The goal of this project was to attempt to create a working recommendation system to provide the probability of a user purchasing new items.

The data mining methods needed to complete this project includes finding the frequent itemsets to get confidence rates (Apriori) and creating a classifier that can accurately predict if a user would re-purchase an item. Combined, the results will be used to calculate purchase probability for new items.

The main motivation for this project was combining algorithms learned in previous homework assignments to complete one task, basically a capstone project for class. Also, I wanted to expand the scope of the Kaggle data set problem by introducing other algorithms to discover new information.

The main challenge of these tasks was efficiently processing large amounts of data. With around 30 million individual purchases, optimizing algorithms and parameters was very important, especially for the Apriori algorithm. The second challenge was generating a large enough feature set to get highly accurate predictions.

Overall, the project was a success with product purchase probabilities up to 30% and re-order prediction accuracy of 90% for both Random forest and Gradient boosting.

2 Data Mining Task

This projects involved 3 parts, finding the frequent itemsets and saving the confidence to a data frame, predicting if a user will re-purchase an item, and combining the results to predict the purchase probability of a user purchasing new items. The input data includes userId, productId, orderId, and information about a users' orders such as if the product has been repurchased before and cart position of the item. The end result is a table of items and their purchase probability for a user/order.

Data mining questions:
- Apriori frequent itemsets
- Random Forest classifier
- Gradient Boosting classifier
- Recommendation system

The key challenges include optimizing algorithms and parameters to handle large amounts of data, feature selection/engineering, and calculating the weights for the recommendation system.

3 Technical Approach

The first step was to get a list of products for each order and pass it into an Apriori algorithm to get the frequent itemsets. The second step was to generate a set of features for training and testing a binary classifier. Research showed that a decision tree based model such as Random Forest or Gradient Boosting would provide the best results. Lastly, the confidence from the frequent itemsets will be multiplied by the accuracy or error rate to return a table of item purchase probabilities.

The challenges were addressed by increase the support rate for the Apriori algorithm and reducing the number of users checked for the recommendation system. This was a trade-off between receiving more information and reducing time complexity. Other considerations include using an IDE that supports parallel processing or running the code on an AWS Spot instance to unlock more processing power.

Both the Apriori and Classifier algorithms use python packages so the only modifications involved data processing and parameter tuning. The algorithm for finding purchase probability is as follows:

1. For every item in an order or user purchase history pass the data into the XGBoost (Gradient Booster) classfier to get the predictions of if an item will be purchased again.

2. Pass each of the products into a dictionary with a value of the accuracy of if the product will be purchased again. For example if item 1 was predicted as 1, the accuracy would be 0.90 otherwise, 0.10 (representing the error rate)

3. For every item of the left hand side (lhs) of the frequent itemset (confidence is defined as what percentage of baskets that have item1 also have item2) multiply by the products accuracy weights. If there are more than one item on

the lhs, find the average accuracy weight.

4. Purchase probability for item2 is equal to the confidence multiplied by the accuracy weight of item1.

5. Other modifications include ensuring all items on the lhs are in the user's order and item 2 is not in the order.

4 Evaluation Methodology

The dataset comes from a Kaggle competition where the goal was to predict if an user will repurchase an item. Although the data was already formatted well, slight modifications was needed to create the frequent itemset table.

For the re-order prediction part of the project, three algorithms were tested, Random Forest, XGBoost, and GradientBoostingClassifier from sklearn. The combined order data was split into a training and testing set and used on the algorithms. This was the main metric needed for the recommendation system and was also the only metric that could be improved with algorithm modifications and feature engineering. Looking at examples of other binary classifier projects, I saw that gradient boosting usually had the best results so I chose to use this algorithm as well as the similar random forest classifier.

5 Results and Discussion

The most common items is a basket are fruits and vegetables, in fact 9 of the top 10 purchased items are produce. This gives us an idea of what products will be in the frequent itemsets.

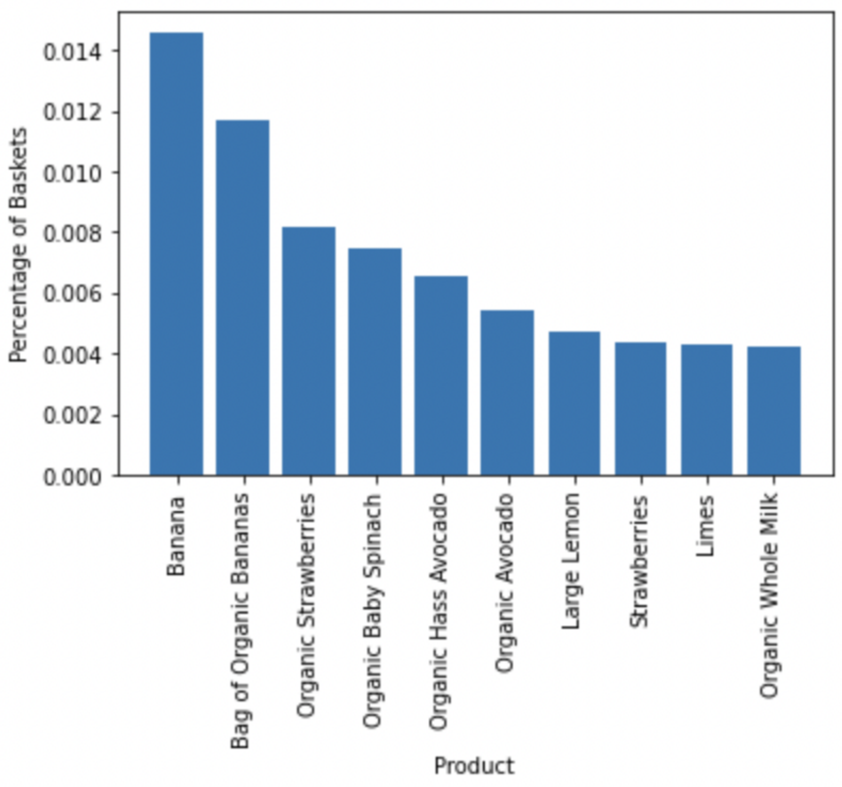
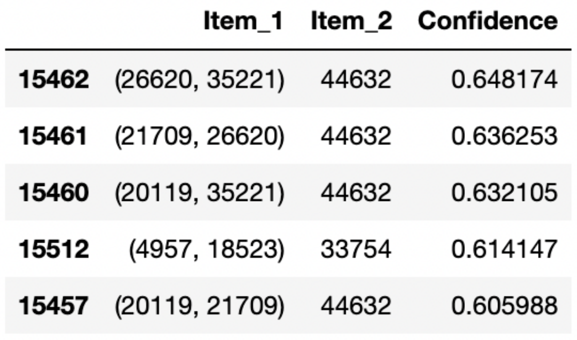


Figure 1: Most popular items as percentage of basket observations



	Item_1	Item_2	Confidence
15462	(26620, 35221)	44632	0.648174
15461	(21709, 26620)	44632	0.636253
15460	(20119, 35221)	44632	0.632105
15512	(4957, 18523)	33754	0.614147
15457	(20119, 21709)	44632	0.605988

Figure 2: Top 5 confidence scores

Figure 2 shows the results from the Apriori algorithm with support of 0.0005, which is around 1,600 of the 3,214,874 baskets. The confidence represents the probability of seeing item 2, given that the basket includes item 1.

The accuracy for all three models were mostly the same but XGBoost was chosen due to its much faster training speed which was important when processing millions of rows. All of the features used and their importance in the XGBoost model can be seen in figure 3. Overall the classifier was successful. However, this may be due to the distribution of the data (figure 4) where the vast majority were predicted as 'Not Reordered' explaining the high accuracy and minimal differences between classifier accuracy.

Table 1: Prediction Results

Model	**Test Accuracy**
GradientBoostingClassifier	0.907136
Random Forest	0.907348
XGBoost	0.908406

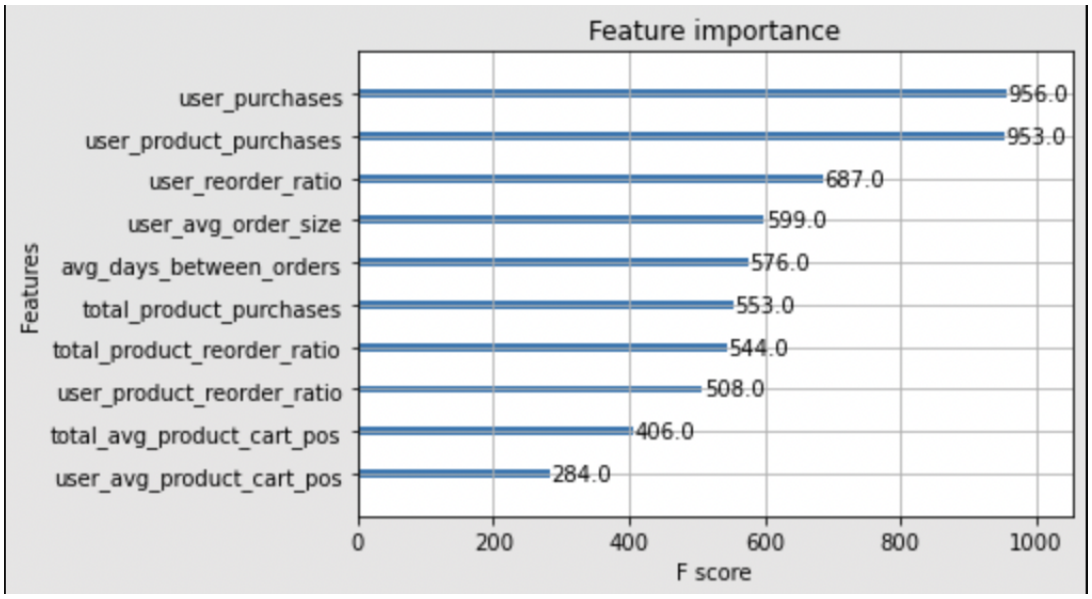


Figure 3: Most important features from XGBoost model

For the most part, the purchase probabilities were very low. This is due to how most items are not reordered as seen in figure 4.

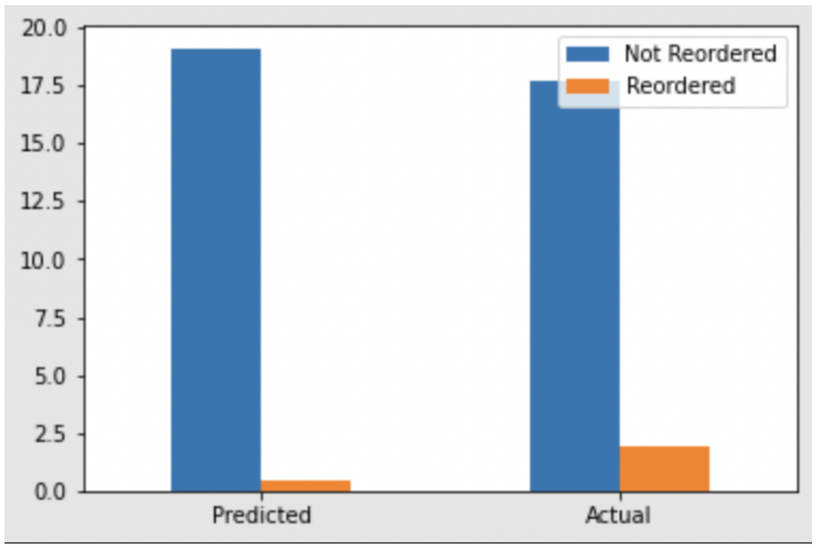
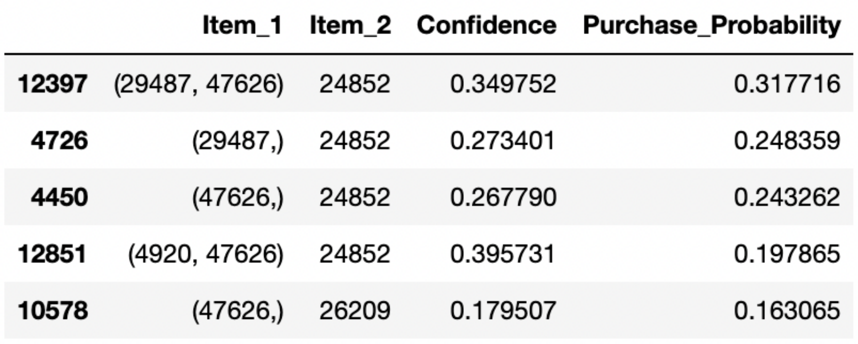


Figure 4: Distribution of average number of items reordered by user

Overall, the recommendation system does work with a few items having high purchase probabilities. However, shown in figure 5, item 2 for the most rows represents Bananas, the most purchased item (figure 1). This indicates bias towards popular products for items with high purchase probabilities.



	Item_1	Item_2	Confidence	Purchase_Probability
12397	(29487, 47626)	24852	0.349752	0.317716
4726	(29487,)	24852	0.273401	0.248359
4450	(47626,)	24852	0.267790	0.243262
12851	(4920, 47626)	24852	0.395731	0.197865
10578	(47626,)	26209	0.179507	0.163065

Figure 5: Example of recommendation output for a user

6 Lessons Learned

A lot was learned with this project, most notably, combining the results of multiple data mining algorithms to provide new information and support better decision making. In this case, the recommendation system was able to take two algorithms that didn't provide much actionable insight to provide user specific product purchase likelihoods.

There is a lot of room for improvement in this project including lowering the Apriori support, more feature engineering, and expanding the recommendation system. With more time and processing power, the support for the Apriori can be reduced, further expanding the frequent itemset table and providing more confidence rates for more products. There wasn't much hyper parameter tuning or cross validation done for the classifiers as the accuracy was very good. Creating more features and tuning the classification models could result in a small increase in test accuracy. Lastly, there is a lot more that can go into the recommendation system part of the project. Some examples include using similar items by department or aisle for non-frequent items or popular items for users with few purchases, both solutions to a cold start problem. Other data such as time of year to recommend seasonal items and even a dataset of common recipes that include a purchased item can provide better recommendations.

7 Acknowledgements and Resources

Datasets: https://www.kaggle.com/competitions/instacart-market-basket-analysis/overview

Apriori algorithm: https://github.com/tommyod/Efficient-Apriori

Gradient Boosting and Random Forest:

https://machinelearningmastery.com/gradient-boosting-with-scikit-learn-xgboost-lightgbm-and-catboost/

https://www.analyticsvidhya.com/blog/2021/09/gradient-boosting-algorithm-a-complete-guide-for-beginners/

https://scikit-learn.org/stable/modules/ensemble.htmlgradient-tree-boosting

8 Code

```
import numpy as np
import pandas as pd
from efficient_apriori import apriori
from sklearn.model_selection import train_test_split as sklearn_train_test_split
from sklearn.metrics import accuracy_score
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from xgboost import plot_importance
import matplotlib.pyplot as plt
```

```
from collections import Counter

def generate_frequent_itemsets(baskets, support=0.0005, confidence=0):
    itemsets, rules = apriori(baskets, min_support=0.0005, min_confidence=0)
    lhs = []
    rhs = []
    conf = []
    lhs_count = max(itemsets.keys()) #Maximum number of items on left hand side
    for i in range(1,lhs_count+1):
        rules_rhs = filter(lambda rule: len(rule.lhs) == i and len(rule.rhs) == 1, rules)
        for rule in sorted(rules_rhs, key=lambda rule: rule.lift):
            lhs.append(rule.lhs)
            rhs.append(rule.rhs[0])
            conf.append(float(rule.count_full/rule.count_lhs))
    frequent_items = pd.DataFrame(
    {'Item_1': lhs,
     'Item_2': rhs,
     'Confidence': conf
    })
    return frequent_items

def product_feature_set(total_orders):
    product_purchases = total_orders.groupby('product_id')['order_id'].count().to_frame('tot
    product_reorder_ratio = total_orders.groupby('product_id')['reordered'].mean().to_frame(
    product_cart_ranking = total_orders.groupby('product_id')['add_to_cart_order'].mean().to
    product_features = product_purchases.merge(product_reorder_ratio, on='product_id', how=
    product_features = product_features.merge(product_cart_ranking, on='product_id', how='le
    return product_features

def user_feature_set(total_orders):
    user_purchases = total_orders.groupby('user_id')['order_number'].max().to_frame('user_pu
    user_reorder_ratio = total_orders.groupby('user_id')['reordered'].mean().to_frame('user_
    user_cart_size = total_orders.groupby(['user_id', 'order_id'])['add_to_cart_order'].max(
    days_between_orders = total_orders.groupby(['user_id', 'order_id'])['days_since_prior_or
    user_features = user_purchases.merge(user_reorder_ratio, on='user_id', how='left')
    user_features = user_features.merge(user_cart_size, on='user_id', how='left')
    user_features = user_features.merge(days_between_orders, on='user_id', how='left')
    return user_features

def user_product_feature_set(total_orders):
    user_product_purchases = total_orders.groupby(['user_id', 'product_id'])['order_number']
    user_product_reorder_ratio = total_orders.groupby(['user_id', 'product_id'])['reordered'
    user_product_avg_cart_pos = total_orders.groupby(['user_id','product_id'])['add_to_cart_
    user_product_features = user_product_purchases.merge(user_product_reorder_ratio, on=['us
    user_product_features = user_product_features.merge(user_product_avg_cart_pos, on=['user
    return user_product_features
```

```
def full_feature_set(product_features, user_features, user_product_features):
    feature_set = user_product_features.merge(user_features, on='user_id', how='outer')
    feature_set = feature_set.merge(product_features, on='product_id', how='outer')
    return feature_set

def train_test_split(orders, feature_set):
    order_types = orders[((orders.eval_set=='train') | (orders.eval_set=='test'))]
    order_types = order_types[['user_id', 'eval_set', 'order_id']]
    data = feature_set.merge(order_types, on='user_id', how='left')
    #training data
    train = data[data.eval_set=='train']
    train_data = train.merge(orders_train[['product_id', 'order_id', 'reordered']], on=['pro
    train_data['reordered'] = train_data['reordered'].fillna(0)
    train_data = train_data.set_index(['user_id', 'product_id'])
    train_data = train_data.drop(['eval_set', 'order_id'], axis = 1)
    #testing data (for Kaggle evaluation)
    test_data = data[data.eval_set=='test']
    test_data = test_data.drop(['eval_set', 'order_id'], axis = 1)
    #test_data = test_data.set_index(['user_id', 'product_id'])
    #Model test/train split
    X_train, X_val, y_train, y_val = sklearn_train_test_split(
        train_data.drop(['reordered'], axis = 1),
        train_data.reordered,
        test_size=0.3,
        random_state=42)
    return X_train, X_val, y_train, y_val, test_data, train_data

def train_classifier(X_train, X_val, y_train, y_val):
    accuracy = {}
    #sklearn gradient booster
    #GBC = GradientBoostingClassifier()
    #GBC.fit(X_train, y_train)
    #GBC_y_pred = GBC.predict(X_val)
    #accuracy['GradientBoostingClassifier'] = accuracy_score(y_val, GBC_y_pred) #0.907136

    #Random forest
    #RF = RandomForestClassifier()
    #RF.fit(X_train, y_train)
    #RF_y_pred = RF.predict(X_val)
    #accuracy['RandomForest'] = accuracy_score(y_val, RF_y_pred) #0.907348

    #XGBoost
    XGB = XGBClassifier()
    XGB.fit(X_train, y_train)
    XGB_y_pred = XGB.predict(X_val)
    accuracy['XGBoost'] = accuracy_score(y_val, XGB_y_pred) #0.908406

    return XGB, accuracy

#get the predictions for every product for a user/order
def get_predictions(df, model, accuracy):
    predictions_dict = {}
    predictions = model.predict(df)
    i = 0
    for index, row in df.iterrows():
        product_id = index[1]
        predictions_dict[product_id] = predictions[i]
        i+=1
    #Saves the prediction as a weight (if reordered, save the accuracy (0.908) else 1-0.908)
    for key in predictions_dict:
        if predictions_dict[key] == 1:
            predictions_dict[key] = a
        else:
            predictions_dict[key] = (1-a)
    return predictions_dict

#For every product in the lhs of the frequent itemsets, multiply by the prediction weights
def calculate_purchase_probability(row):
    item1 = row["Item_1"]
    confidence = row["Confidence"]

    if len(item1) == 1:
        constant = predictions.get(item1[0], 0)
        purchase_probability = confidence * constant
    else:
        #Calculate average weight of the item1 basket if there is more than one item
        total_weight = 0
        for item in item1:
            weight = predictions.get(item, 0)
            total_weight += weight
        average_weight = total_weight / len(item1)
        purchase_probability = confidence * average_weight

    return purchase_probability

#Calculate purchase probability by multiplying confidence of items the order includes by the
#Only returns items not purchased and order must include all items on lhs if more than one.
def get_recommendations(user, xgb, a):
    prediction_data = feature_set.loc[feature_set.user_id == user]
    prediction_data = prediction_data.set_index(['user_id', 'product_id'])
    global predictions
    predictions = get_predictions(prediction_data, xgb, a)

    recommendations = frequent_items.copy()
    recommendations["Purchase_Probability"] = recommendations.apply(calculate_purchase_proba
    recommendations = recommendations[recommendations['Item_1'].apply(lambda x: all(item in
    recommendations = recommendations[~recommendations['Item_2'].isin(predictions.keys())] #
    recommendations = recommendations.sort_values("Purchase_Probability", ascending=False)
    return recommendations

products = pd.read_csv("instacart-market-basket-analysis/products.csv")
orders = pd.read_csv("instacart-market-basket-analysis/orders.csv")
orders_prior = pd.read_csv("instacart-market-basket-analysis/order_products__prior.csv")
orders_train = pd.read_csv("instacart-market-basket-analysis/order_products__train.csv")
aisles = pd.read_csv("instacart-market-basket-analysis/aisles.csv")
departments = pd.read_csv("instacart-market-basket-analysis/departments.csv")

total_orders = orders.merge(orders_prior, on='order_id', how='inner')

#Get order products list as product name
products_list = orders_prior[["order_id", "product_id"]]
products_list = products_list.merge(products, on='product_id', how='inner')
products_list = products_list.groupby('order_id')['product_name'].apply(list)
products_list = products_list.to_list()
#Get top 10 products as percentage of occurances in orders
product_counts = Counter(item for basket in products_list for item in basket)
total_products = sum(product_counts.values())
product_percentages = {item: count/total_products for item, count in product_counts.items()
sorted_products = sorted(product_percentages.items(), key=lambda x: x[1], reverse=True)
sorted_products = sorted_products[:10]
#Plot the top 10 most purchased products
fig, ax = plt.subplots()
ax.bar([item[0] for item in sorted_products], [item[1] for item in sorted_products])
plt.xticks(rotation=90)
ax.set_xlabel('Product')
ax.set_ylabel('Percentage of Baskets')
plt.show()

#Get order products list as product number
products_list = orders_prior[["order_id", "product_id"]]
products_list = products_list.groupby('order_id')['product_id'].apply(list)
products_list = products_list.to_list()

frequent_items = generate_frequent_itemsets(products_list)

product_features = product_feature_set(total_orders)
```

```
user_features = user_feature_set(total_orders)
user_product_features = user_product_feature_set(total_orders)
feature_set = full_feature_set(product_features, user_features, user_product_features)
X_train, X_val, y_train, y_val, test_data, train_data = train_test_split(orders, feature_se

XGB, accuracy = train_classifier(X_train, X_val, y_train, y_val)

#Plot the distribution of the average number of products reordered by users for both the ac
pred = pd.Series(XGB_y_pred, name='Predicted')
data = pd.DataFrame(y_val).reset_index()
data = pd.concat([data, pred], axis=1)
counts = data.groupby('user_id').agg({'reordered': ['sum', 'count'], 'Predicted': ['sum', '
counts.reset_index(inplace=True)
counts = counts.set_index(['level_0', 'level_1']).unstack()
counts.columns = counts.columns.swaplevel()
counts = counts.sort_index(level=0, axis=1)
counts.index = ['Predicted', 'Actual']
counts.columns = ['Not Reordered', 'Reordered']
counts.iloc[:,0] = counts.iloc[:,0] - counts.iloc[:,1]
counts = counts.round(2)
counts.plot(kind='bar',rot=0)

plot_importance(XGB)
plt.show()

a = accuracy['XGBoost']
frequent_items.sort_values("Confidence", ascending=False).head(5)

#Find an example with the highest purchase probability
users = feature_set['user_id'].unique().tolist()
max_prob = 0
user = 0
for i in range(0,1000):
    recommendations = get_recommendations(users[i], XGB, a)
    if not recommendations.empty:
        if (recommendations.iloc[0]['Purchase_Probability'] > max_prob):
            max_prob = recommendations.iloc[0]['Purchase_Probability']
            user = users[i]
recommendations = get_recommendations(user, XGB, a)

recommendations.head(5)
products.loc[products.product_id == 24852]
```