Product Recommendation System Prototype

Group 8-

Craig Atkinson a1669436
Lalitphan Sae-teoh a1932456
Pratham Maharjan a19441800

In [22]:
```
class Recommendations:
    """
    Usage: Product Recommendation System with a Terminal Text Interface
    Input: Text Interface will ask for UserID and "with or without frequent patterns"
    Output: List of product recommendations the provided User
    """

    def __init__(self,data):
        #read the data once and assign to variable
        self.data=data
        #train the AlternatingLeastSquares model once and assign to variable
        self.trained_model=None

    def pattern_mining(self,apriori=False):
        """
        Function Provided by Task 1 Pratham Maharjan.
        """
        from mlxtend.preprocessing import TransactionEncoder
        from mlxtend.frequent_patterns import apriori, association_rules
        from mlxtend.frequent_patterns import fpmax, fpgrowth

        #group columns user and set as list
        grouped = self.data.groupby(['User_id', 'Date'])['itemDescription'].apply(list).reset_index()

        #only the itemDescription column as product transactions list
        transactions = grouped['itemDescription'].tolist()

        #encode transations list as one-Hot vector
        te = TransactionEncoder()
        te_array = te.fit(transactions).transform(transactions)
        df_encoded = pd.DataFrame(te_array, columns=te.columns_)
        #use apriori or fp-grown functions
        if apriori==True:
            #use apriori algor to generation association rules
            frequent_itemsets_apriori = apriori(df_encoded, min_support=0.001, use_colnames=True)
            frequent_itemsets_apriori=frequent_itemsets_apriori.sort_values('support')
            #create association Rules
            rules_apriori = association_rules(frequent_itemsets_apriori, metric="lift", min_threshold=1.2)
            return rules_apriori
        else:
            #use frequent Itemset Mining using FP-growth algor
            frequent_itemsets_fp = fpgrowth(df_encoded, min_support=0.001, use_colnames=True)
            rules_fp = association_rules(frequent_itemsets_fp, metric="lift", min_threshold=1.2)
            return rules_fp


    def collaborative_filtering(self,user_id, use_patterns):
        """
        Function Provided by Task 2 Lalitphan Sae-teoh
        """
        import numpy as np
        from collections import defaultdict
        from scipy.sparse import csr_matrix
        from implicit.als import AlternatingLeastSquares
        from implicit.nearest_neighbours import bm25_weight

        def recommend_without_freqset(user_items_matrix, user_id, user_mapping, item_mapping, number_of_items=5)

            # user_id index
            user_index = user_mapping.index(user_id)
            recommended = model.recommend(user_index, sparse_matrix[user_index], N=number_of_items)

            recommended_items = [item_mapping[item_id] for item_id in recommended[0]]
            recommended_score = recommended[1]

            return recommended_items
```

```
        def recommend_with_freqset(user_items_matrix, user_id, rules, metrics, user_mapping, item_mapping, numbe
            # Get items the user recently interacted with
            user_purchased_items = set(df[df['User_id'] == user_id]['itemDescription'].unique().tolist())
            rule_boost = defaultdict(float)
            for _, row in rules.iterrows():
                if set(row['antecedents']).issubset(set(user_purchased_items)):
                    for item in row['consequents']:
                        rule_boost[item] += row[metrics]

            user_index = user_mapping.index(user_id)

            recommended = model.recommend(user_index, sparse_matrix[user_index], N=20)
            recommended_top_items = [(item_id, item_mapping[item_id]) for item_id in recommended[0]]
            recommended_score = recommended[1]

            final_scores = {}
            for index, item in enumerate(recommended_top_items):
                item_name = item[1]
                boost = rule_boost.get(item_name, 0)
                final_scores[item_name] = recommended_score[index] + boost

            final_recommendations = sorted(final_scores.items(), key=lambda x: -x[1])[:5]
            final_recommendation_items = []

            for item_name, score in final_recommendations:
                final_recommendation_items.append(item_name)

            return final_recommendation_items

        #data preprocessing
        df=self.data
        df['days_ago'] = (df.index.max() - df.index)
        df['days_ago'] = df['days_ago'].dt.components.days
        #4 months recency bias
        df['recency_weight'] = np.exp(-df['days_ago'] / 120)


        df_user_items = df.groupby(['User_id', 'itemDescription']).agg({'Date':['count']}).reset_index()
        df_user_items.columns = ['User_id', 'itemDescription', 'frequency']

        #Sum weighted counts per user-item
        weighted_df = df.groupby(['User_id', 'itemDescription'])['recency_weight'].sum().reset_index()

        #multiply frequency × mean recency weight per user-item
        df_user_items_recency = df_user_items.merge(weighted_df, on=['User_id', 'itemDescription'])
        df_user_items_recency['final_weight'] = df_user_items_recency['frequency'] * df_user_items_recency['rece
        user_items_matrix = df_user_items_recency.pivot(index="User_id", columns="itemDescription", values="fina
        user_items_matrix.fillna(0, inplace=True)

        #create sparse matrix from user-item matrix and apply BM25 weighing
        sparse_matrix = csr_matrix(user_items_matrix.values)
        weighted_matrix = bm25_weight(sparse_matrix.T).T

        #train the model, first use default parameters (should do parameter tuning in later versions)
        if self.trained_model==None:
            model = AlternatingLeastSquares(factors=50, regularization=0.01, iterations=30, random_state=8)
            model.fit(weighted_matrix,show_progress=0)
        else:
            model=self.trained_model

        #map the provided user id
        user_mapping = list(user_items_matrix.index)
        user_index = user_mapping.index(user_id)
        item_mapping = list(user_items_matrix.columns)

        #get the recommended items
        if use_patterns == 'with':
            #with pattern mining rules
            rules= self.pattern_mining()
            return recommend_with_freqset(user_items_matrix, user_id, rules=rules, metrics='confidence',user_mapp
        else:
            #without pattern mining rules
            return recommend_without_freqset(user_items_matrix, user_id, user_mapping=user_mapping, item_mapping=

    def text_program(self,user=None):
        """
        Task 3: Runs the Text Interface, which calls the other tasks
        """

        while True:
```

```
            if user==None:
                user_id_input = input("Enter user ID or enter 'q' to quit:\n")
                if user_id_input.lower() == 'q':
                    break
                #validate input is numerical
                try:
                    user_id = int(user_id_input)
                except:
                    print("Please Only Enter a Numerical User ID")
                    continue

                #for a new user, recommend the 10 most popular items
                if user_id not in self.data['User_id'].unique():
                    print("New User Detected, Not in Dataset. Best Selling items:")
                    top_items = self.data['itemDescription'].value_counts().head(5).index.tolist()
                    for item in top_items:
                        print(item)
                    print("\n------------------------------\n")
                    continue

                method_input = input('With or Without frequent patterns? enter "with" or "without":\n')
                use_patterns = method_input.lower()
            else:
                user_id = user
                use_patterns='without'

            if use_patterns in ['with', 'without']:
                recommendations = self.collaborative_filtering(user_id, use_patterns)
                print("\nProduct Recommendations for user: " + str(user_id))
                for item in recommendations:
                    print(item)
                print("\n------------------------------\n")
            else:
                print('Please Only Enter "with" or "without" ')

"""
Main Function, Loads the training dataset and starts the Recommendation System Interface.
"""
if __name__ == "__main__":
    import pandas as pd
    import warnings
    warnings.filterwarnings('ignore')

    #laod the data
    data = pd.read_csv('./data/Groceries data train.csv')
    data = data.dropna()

    #create pandas datetime index
    data['datetime'] = pd.to_datetime(data['Date'], dayfirst=True).dt.date
    data.set_index('datetime', inplace=True)

    print("Welcome to the Product Recommendation System")
    recommend = Recommendations(data)
    #run the program
    recommend.text_program()
```

```
Welcome to the Product Recommendation System
Please Only Enter a Numerical User ID
New User Detected, Not in Dataset. Best Selling items:
whole milk
other vegetables
rolls/buns
soda
yogurt

------------------------------

Product Recommendations for user: 3000
dishes
meat
spread cheese
frozen meals
seasonal products

------------------------------
```

```
Product Recommendations for user: 3000
dishes
meat
spread cheese
frozen meals
seasonal products

------------------------------

New User Detected, Not in Dataset. Best Selling items:
whole milk
other vegetables
rolls/buns
soda
yogurt

------------------------------
```