Table of Contents

Recommending Products

[1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics.pairwise import cosine_similarity

from sklearn.model_selection import train_test_split

from sklearn.neighbors.kdtree import KDTree
```

Importing Dataset

[2]:
```
#Read the data set
df = pd.read_csv('/content/drive/MyDrive/Amazon_Electronics.csv', header=None)
df.columns = ['user_id', 'product_id', 'rating', 'timestamp']

df = df[df['timestamp', ].notnull()]

df_copy = df.copy(deep=True)
```

EDA - Exploratory Data Analysis

Check for :

- Shape
- Datatypes
- Missing values

Finding get the summary and check

- rating distribution
- Number of users and products
- Users with highest no of ratings

Shape

[3]:
```
print('Rows, columns: {}'.format(
print('No of rows: {}'.format(
print('No of columns: {}'.format(
print('Shape: {}'.format(
df.shape
```

Datatypes

[4]:
```
df.info()
```

Missing value analysis

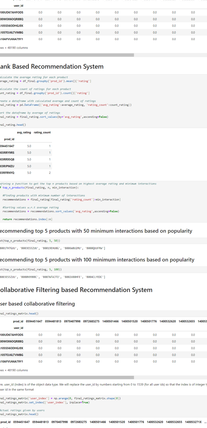
[5]:
```
# Find number of missing values in each column
df.isnull().sum()
```

Summary

[6]:
```
# Summary statistics of 'rating' variable
df['rating'].describe()
```

Rating distribution

[7]:
```
#Rating distribution
df['rating'].value_counts().sort(ascending=False)
```



The distribution is skewed to the right. Over 50% of the ratings are 5, followed by a little below 25% with 4 star ratings. And the percentages of ratings keep going down until below 5% of the ratings are of 1 star.

No of unique users and items

[8]:
```
# Number of unique users and items in the data
print('Number of unique users in the data = ', df['user_id'].nunique())
print('Number of unique items in the data = ', df['product_id'].nunique())
```

Users with most no of rating

[9]:
```
# Top 10 users based on rating
most_rated = df.groupby('user_id').size().sort_values(ascending=False)[:10]
most_rated
```

Pre-Processing

[10]:
```
# Number of unique user id and product id in the data
print('Number of unique users in the data = ', df['user_id'].nunique())
print('Number of unique products in the data = ', df['product_id'].nunique())

# Number of ratings per user
print('Number of ratings per user -> Mean = ', df['user_id'].value_counts().mean())
print('Number of ratings per user -> Median = ', df['user_id'].value_counts().median())

# Number of ratings per product
print('Number of ratings per product -> Mean = ', df['product_id'].value_counts().mean())
print('Number of ratings per product -> Median = ', df['product_id'].value_counts().median())
```

Checking the sparsity of the rating matrix

[11]:
```
# Matrix with one row for each user and one column for each product
pivot_df = df.pivot(index='user_id', columns='product_id', values='rating').fillna(0)

pivot_df.head(5).T.head(5)
```

Rank Based Recommendation System

[12]:
```
# Calculating the average rating for each product
average_rating = df.groupby(['product_id'],)['rating'].

# Calculating the count of ratings for each product
count_rating = df.groupby(['product_id'],)['rating'].

# Create a dataframe with calculated average and count of ratings
final_rating = pd.DataFrame(zip(average_rating.index, average_rating, count_rating),

# Find the maximum no ratings of ratings
final_rating[final_rating.sort_values(['rating_count'],ascending=False)]

final_rating.head()
```

[13]:
```
# Finding products with minimum number of ratings
recommendations = final_rating[final_rating['rating_count'] >min_interaction]

# Sorting values w.r.t average rating
recommendations = recommendations.sort_values( ag_rating ,ascending=False)

return recommendations.index[:n]
```

Recommending top 5 products with 50 minimum interactions based on popularity

[14]:
```
find_top_n_products(final_rating, 5, 50)
```

Recommending top 5 products with 100 minimum interactions based on popularity

[15]:
```
find_top_n_products(final_rating, 5, 100)
```

Collaborative Filtering based Recommendation System

User based collaborative filtering

[16]:
```
# Find similarity matrix for user
pivot_df.T.fillna(0).head(5).T
```

[17]:
```
# Matrix with one row for each user and one column for each product
pivot_matrix = pivot_df.values

pivot_matrix.shape
```

[18]:
```
# Finding similarity between first user and all other users
user_similarity = cosine_similarity(pivot_matrix[0].reshape(1, -1), pivot_matrix)[0].

# Number of users
len(user_similarity)
```

[19]:
```
# Finding top 10 similar users to the user index 3 and their similarity score
similar_indices = user_similarity[user_similarity.argsort()[::-1][1:11]]

similar_indices
```

[20]:
```
# Finding top 10 similar users to the user index 1521 and their similarity score
similar = cosine_similarity(pivot_matrix[1521].reshape(1,-1), pivot_matrix)[0].

similar
```

[21]:
```
# Finding the similarity score
similar_users[1521,find_similar_users(1521)[0:10]]
```

[22]:
```
# Defining the recommendation function to get recommendations by using the similar users' preferences
def recommend(user_index, num_of_products, interactions_matrix):

    # Calculating cosine similarity between the user and each other user
    similarity = cosine_similarity(interactions_matrix[user_index],interactions_matrix).[0].

    # Finding the most similar user to the user index
    similar_indices = similarity[similarity.argsort()[::-1][1:]]

    # Finding the products which are not rated by the user
    interactions = interactions_matrix[user_index].

    for product in range(len(interactions[:num_of_products*5*10*10*10
```