Food Recommendation System

CSD 311 Introduction to Machine Learning

Made By

Jaskaran Singh Gujral (1810110091)

Namrata Raina (1810110138)

S Sanjeeth Chandhira Rajeshwaran (1810110195)

Nikhil Khandelwal (1810110293)

We shall not be using any libraries in our approaches but the read_csv function is required to read the CSV file containing the dataset and hence cannot be avoided. ¶

In [1]:

```
from pandas import read_csv
```

Reading the data from the CSV file into a DataFrame

It should be noted here that the dataset has been created by us ourselves but that particular code has been excluded from this Jupyter Notebook since it is not part of the problem statement.

In [2]:

```
data = read_csv(r'Data.csv')
data.head()
```

Out[2]:

	ID	Cuisine	Dish	Rating	Value
0	71	Dessert	Rasgulla	4	26
1	50	Dessert	Gulab Jamun	5	25
2	72	Fast Food	Subway	2	28
3	64	Indian	Butter Naan	0	3
4	40	Mughlai	Kebab	3	13

The simplest algorithm to recommend a food item based on history is to recommend the item purchased the maximum number of times by the concerned user. The below function returns the same.

In [3]:

```
def maxEaten(ID, data):
    data = data['Dish'][data['ID'] == ID]
    data = data.value_counts()
    data = data.idxmax()

    return data
```

One of the most widely used techniques for recommendation systems is Collaborative Filtering. In this, the algorithm takes into account the similarity between users, and recommends the food item that is ordered by the majority of the users whose eating habits are similar to the concerned user.

In [4]:

```
def collaborativeFiltering(ID, data):
    dishes = data['Dish'][data['ID'] == ID].unique()

    IDs = list()

    for i in data['ID'].unique():
        if set(dishes).issubset(set(data['Dish'][data['ID'] == i].unique())):
            IDs.append(i)

    newData = data[data['ID'].isin(IDs)]

    newDishes = newData['Dish'][-newData['Dish'].isin(dishes)].unique()

    return newData['Dish'][newData['Dish'].isin(newDishes)][newData['ID'].isin(IDs)].va
lue_counts().idxmax()
```

The below algorithm could be called a modified version of KNN. Since it is not in the scope of the project to add the actual recipes on which KNN could be applied, we have added values to each dish based on our knowledge of the recipes. Closer the recipes of each dish, the closer is the value given. Based on these values, an average is taken of the dishes ordered previously by the concerned user. The dish which has the closest value is then recommended to the user.

In [5]:

```
def valueBased(ID, data):
    sumValue = sum(data['Value'][data['ID'] == ID])

    length = len(data['Value'][data['ID'] == ID])

    roundedValue = round(sumValue / length)

    return data['Dish'][data['Value'] == roundedValue].head(1).item()
```

One of the most popular techniques used to recommend anything is based on popularity of the item. However, this method is not personalized to the user. In our case, we have created an algorithm which is somewhat personalized because it takes into account the cuisine that is preferred by the concerned user. Based on this, it recommends the most popular dish in that cuisine segment.

In [6]:

```
def popularityBased(ID, data):
    cuisine = data['Cuisine'][data['ID'] == ID]
    cuisine = cuisine.value_counts()
    cuisine = cuisine.idxmax()

    dish = data[data['Cuisine'] == cuisine]
    dish = dish.groupby('Dish', as_index = False)
    dish = dish['Rating'].mean()
    dish = dish.sort_values('Rating', ascending = False)
    dish = dish.reset_index(drop = True)

    return dish['Dish'].head(1).item()
```

The last approach is based on rating given by the user to each dish he has ordered previously. This approach takes into account the dish which has the highest average rating by the user. Then it looks for users who have given a similar rating to that dish. From the set of these users, it is then found out which dish has the highest average rating. That dish is recommended to the user.

In [7]:

```
def ratingBased(ID, data):
    dish = data[data['ID'] == ID]
    dish = dish.groupby('Dish', as_index = False)
    dish = dish['Rating'].mean()
    dish = dish.sort_values('Rating', ascending = False)
    dish = dish.reset_index(drop = True)

    rating = dish['Rating'].head(1).item()
    dish = dish['Dish'].head(1).item()

    IDs = data['ID'][data['Dish'] == dish][data['Rating'] >= rating - 1][data['Rating']
<= rating + 1]

    newData = data[data['ID'].isin(IDs)]
    newData = newData.groupby('Dish', as_index = False)
    newData = newData['Rating'].mean()
    newData = newData.sort_values('Rating', ascending = False)
    newData = newData.reset_index(drop = True)

    return newData['Dish'].head(1).item()
```

The main function is used to combine all of the methods given above.

In [8]:

```
def main(ID, data):
    print("Recommendation based on maximum number of dishes eaten: ", maxEaten(ID, data
))
    print("\nRecommendation based on collaborative filtering: ", collaborativeFiltering
(ID, data))
    print("\nRecommendation based on distance of values: ", valueBased(ID, data))
    print("\nRecommendation based on popularity: ", popularityBased(ID, data))
    print("\nRecommendation based on rating: ", ratingBased(ID, data))
```

In the saved instance of the Jupter Notebook, the ID (input value) is taken as 13.

In [9]:

```
main(int(input()), data)
```

```
Recommendation based on maximum number of dishes eaten:  Cake

Recommendation based on collaborative filtering:  Chowmein

Recommendation based on distance of values:  Chowmein

Recommendation based on popularity:  Gulab Jamun

Recommendation based on rating:  Sambhar Vada
```