p2822004_p2822007

May 23, 2021

0.1 Master of Science in Business Analytics

0.2 Department of Management Science and Technology

0.3 Athens University of Economics and Business

0.4 Mining Big Datasets

0.5 1st Assignment

Konstantina Georgiopoulou (p2822004) Anastasios Theodorou (p2822007)

The goal of this assignment is to calculate the similarity between supermarket customers, using the demographic characteristics of 10.000 customers along with a list of groceries they bought. The workflow used in order to compute his/her 10 most similar customers. Moreover, we implemented a classification algorithm in order to predict his rating to the supermarket.

- Before we proceed to anything we had to import the appropriate libraries and connect to our `google-colab` account.

[1]:
```
#libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import sklearn as sk
import statistics as st
import itertools

%matplotlib inline
```

0.5.1 Question 1: Import and pre-process the dataset with customers

- After that, we downloaded the file `groceries.csv` and we proceeded in the creation of a dataframe.

[2]:
```
groceries=pd.read_csv('groceries.csv', sep=';')
groceries
```

[2]:
```
      Customer_ID  Age     Sex Marital_Status  Education  Income  \
0               1   75    male        married    primary   20000
1               2   61  female         single  secondary   28000
2               3   32    male         single  secondary   34000
3               4   62    male        married    primary   31000
4               5   66  female        married  secondary   19000
…             …   ..     …             …      …     …
9995         9996   54    male        married    primary   15000
9996         9997   73    male       divorced   tertiary   30000
9997         9998   38    male        married    primary   46000
9998         9999   44  female        married  secondary   23000
9999        10000   59    male        married   tertiary   21000

     Customer_Rating  Persons_in_Household     Occupation  \
0          very_good                     3        retired
1               good                     1      housemaid
2          very_good                     1    blue-collar
3          very_good                     3    blue-collar
4               good                     3        retired
…                …                    …            …
9995            good                     3     unemployed
9996            good                     1        retired
9997            fair                     4    blue-collar
9998            poor                     4      housemaid
9999            good                     2  self-employed

                                              Groceries
0     citrus fruit,semi-finished bread,margarine,rea…
1                         tropical fruit,yogurt,coffee
2                                           whole milk
3            pip fruit,yogurt,cream cheese,meat spreads
4     other vegetables,whole milk,condensed milk,lon…
…                                                 …
9995  berries,root vegetables,whole milk,butter,roll…
9996  meat,other vegetables,whole milk,beverages,rol…
9997                                              soda
9998  sausage,citrus fruit,tropical fruit,pip fruit,…
9999  soft cheese,cream cheese,rolls/buns,soda,speci…

[10000 rows x 10 columns]
```

- We saved the average of the columns `Age` and `Income` in two variables in order to use them later. For that computation we had to use only those cells that did not contain missing values.

[3]:
```python
avg_age = str(int(divmod(np.asarray(groceries[groceries['Age'].str.isspace() ==␣
↪False]['Age'], dtype=np.int).mean(),1)[0]))
avg_inc = str(int(divmod(np.asarray(groceries[groceries['Income'].str.isspace()␣
↪== False]['Income'], dtype=np.int).mean(),1)[0]))
```

```
print(avg_age)
avg_inc
```

```
53
```

[3]:
```
'30036'
```

- Later, we converted all the columns into `string` format, so as to make the transforming of the data easier.

[4]:
```
headers = groceries.columns.tolist()
data_types = { header: np.str for header in headers }
data_types
```

[4]:
```
{'Customer_ID': str,
 'Age': str,
 'Sex': str,
 'Marital_Status': str,
 'Education': str,
 'Income': str,
 'Customer_Rating': str,
 'Persons_in_Household': str,
 'Occupation': str,
 'Groceries': str}
```

[5]:
```
groceries = groceries.astype(data_types)
groceries.head()
```

[5]:
```
  Customer_ID Age     Sex Marital_Status  Education Income Customer_Rating  \
0           1  75    male        married    primary  20000       very_good
1           2  61  female         single  secondary  28000            good
2           3  32    male         single  secondary  34000       very_good
3           4  62    male        married    primary  31000       very_good
4           5  66  female        married  secondary  19000            good

  Persons_in_Household   Occupation  \
0                    3      retired
1                    1    housemaid
2                    1  blue-collar
3                    3  blue-collar
4                    3      retired

                                           Groceries
0  citrus fruit,semi-finished bread,margarine,rea…
1                     tropical fruit,yogurt,coffee
2                                         whole milk
3        pip fruit,yogurt,cream cheese,meat spreads
4  other vegetables,whole milk,condensed milk,lon…
```

- In order to fill the missing values it was necessary to locate them and replace the white-spaces with the means we found earlier. So, after that we had only to convert the columns into `integer` type and set the variable `Customer_ID` as index of our final data-frame.

[6]:
```
#fill nas
groceries['Age'] = groceries['Age'].str.replace(' ', avg_age, regex=False).
↪astype('int64')
groceries['Income'] = groceries['Income'].str.replace(' ', avg_inc,␣
↪regex=False).astype('int64')
groceries['Customer_ID'] = groceries['Customer_ID'].astype('int64')
groceries['Persons_in_Household'] = groceries['Persons_in_Household'].
↪astype('int64')
groceries = groceries.set_index('Customer_ID')
groceries.head()
```

[6]:
```
             Age     Sex Marital_Status  Education  Income Customer_Rating  \
Customer_ID
1             75    male        married    primary   20000       very_good
2             61  female         single  secondary   28000            good
3             32    male         single  secondary   34000       very_good
4             62    male        married    primary   31000       very_good
5             66  female        married  secondary   19000            good

             Persons_in_Household   Occupation  \
Customer_ID
1                               3      retired
2                               1    housemaid
3                               1  blue-collar
4                               3  blue-collar
5                               3      retired

                                                     Groceries
Customer_ID
1            citrus fruit,semi-finished bread,margarine,rea…
2                                tropical fruit,yogurt,coffee
3                                                  whole milk
4                   pip fruit,yogurt,cream cheese,meat spreads
5            other vegetables,whole milk,condensed milk,lon…
```

- Check that everything gone well:

[7]:
```
groceries.isnull().sum()
```

[7]:
```
Age                     0
Sex                     0
Marital_Status          0
Education               0
Income                  0
```

```
Customer_Rating         0
Persons_in_Household    0
Occupation              0
Groceries               0
dtype: int64
```

0.5.2 Question 2: Compute data (dis-)similarity

We have 4 types of data and for each one of them we computed the similarity according to the following types.

- **numeric_dissimilarity:**

$$d(a, b) = |a - b|/maxvalue - minvalue$$

- **nominal_dissimilarity:**
$$d(a, b) = 0 if a = b, 1 otherwise$$

- **ordinal_dissimilarity:**

$$d(a, b) = |rank(a) - rank(b)|/maxrank - maxrank$$

- **set_dissimilarity:** (using Jaccard similarity)

$$1 - intersection/union$$

  Types of Data

  - **Numeric:** Age, Income, Person in Household
  - **Nominal:** Sex, Marital Status, Occupation
  - **Ordinal:** Education, Customer Rating
  - **Set:** Groceries

- We could create an `m*m` dissimilarity matrices, where `m` is the number of observations(m=10.000) for each of the variables (we have 9 variables), but this is very resource-intensive, so we decided to do the computations on-the-fly, for specific pairs of customers.

- According to all of these, we created a function called `dissimilarityFunction` in which we compute the dissimilarity of each attribute of every pair of customers.

- So, first, we convert the ordinal variables into numeric.

[8]:
```
#map ordinal variables
print(groceries.Education.unique())
print(groceries.Customer_Rating.unique())
mapping1 = {'poor': 1, 'fair': 2, 'good': 3, 'very_good': 4, 'excellent': 5}
mapping2 = {'primary': 1, 'secondary': 2, 'tertiary': 3}
groceries['Customer_Rating'] = groceries['Customer_Rating'].map(mapping1)
groceries['Education'] = groceries['Education'].map(mapping2)
groceries.head()
```

```
['primary' 'secondary' 'tertiary']
['very_good' 'good' 'fair' 'excellent' 'poor']
```

```
[8]:              Age     Sex Marital_Status  Education  Income  Customer_Rating  \
Customer_ID
1             75    male        married          1   20000                4
2             61  female         single          2   28000                3
3             32    male         single          2   34000                4
4             62    male        married          1   31000                4
5             66  female        married          2   19000                3

             Persons_in_Household   Occupation  \
Customer_ID
1                               3      retired
2                               1    housemaid
3                               1  blue-collar
4                               3  blue-collar
5                               3      retired

                                                     Groceries
Customer_ID
1            citrus fruit,semi-finished bread,margarine,rea…
2                                tropical fruit,yogurt,coffee
3                                                   whole milk
4                    pip fruit,yogurt,cream cheese,meat spreads
5            other vegetables,whole milk,condensed milk,lon…
```

```
[9]: groceries = groceries.reset_index()
```

- Next, we create the function in which we calculate the dissimilarity, according to the previous types. This function takes as input the `ids` of 2 customers and returns each dissimilarity.

```
[10]: #dissimilarityFunction
def dissimilarityFunction(a, b):
    # Age
    age1 = groceries['Age'].loc[groceries['Customer_ID'] == a].values[0]
    age2 = groceries['Age'].loc[groceries['Customer_ID'] == b].values[0]
    AgeDissimilarity=(abs(age1 - age2)) /(max(groceries['Age']) -␣
 ↪min(groceries['Age']))

    #sex
    sexa = groceries['Sex'].loc[groceries['Customer_ID'] == a].values[0]
    sexb = groceries['Sex'].loc[groceries['Customer_ID'] == b].values[0]
    if (sexa==sexb):
        SexSimilarity = 1
    else:
        SexSimilarity = 0

    SexDissimilarity = 1 - SexSimilarity

    #martital
```

```
    martitala = groceries['Marital_Status'].loc[groceries['Customer_ID'] == a].
↪values[0]
    martitalb = groceries['Marital_Status'].loc[groceries['Customer_ID'] == b].
↪values[0]
    if (martitala==martitalb):
        MartitalSimilarity = 1
    else:
        MartitalSimilarity= 0

    MartitalDissimilarity = 1 - MartitalSimilarity

    # Education
    educationa = groceries['Education'].loc[groceries['Customer_ID'] == a].
↪values[0]
    educationb = groceries['Education'].loc[groceries['Customer_ID'] == b].
↪values[0]

    EducationDissimilarity= abs(educationa - educationb) /␣
↪(max(groceries['Education']) - min(groceries['Education']))

    #income
    incomea = groceries['Income'].loc[groceries['Customer_ID'] == a].values[0]
    incomeb = groceries['Income'].loc[groceries['Customer_ID'] == b].values[0]
    IncomeDissimilarity=(abs(incomea - incomeb)) /(max(groceries['Income']) -␣
↪min(groceries['Income']))

    #customer rating
    ratinga = groceries['Customer_Rating'].loc[groceries['Customer_ID'] == a].
↪values[0]
    ratingb = groceries['Customer_Rating'].loc[groceries['Customer_ID'] == b].
↪values[0]

    RatingDissimilarity= abs(ratinga - ratingb) /␣
↪(max(groceries['Customer_Rating']) - min(groceries['Customer_Rating']))

    #Persons_in_Household
    personsa = groceries['Persons_in_Household'].loc[groceries['Customer_ID']␣
↪== a].values[0]
    personsb = groceries['Persons_in_Household'].loc[groceries['Customer_ID']␣
↪== b].values[0]
    Persons_in_HouseholdDissimilarity=(abs(personsa - personsb)) /
↪(max(groceries['Persons_in_Household']) -␣
↪min(groceries['Persons_in_Household']))

    #Occupation
```

```
    occupationa = groceries['Occupation'].loc[groceries['Customer_ID'] == a].
↪values[0]
    occupationb = groceries['Occupation'].loc[groceries['Customer_ID'] == b].
↪values[0]
    if (occupationa==occupationb):
       OccupationSimilarity = 1
    else:
       OccupationSimilarity = 0

    OccupationDissimilarity = 1 - OccupationSimilarity

    #Groceries
    list1= groceries['Groceries'].loc[groceries['Customer_ID'] == a].values[0]
    list2= groceries['Groceries'].loc[groceries['Customer_ID'] == b].values[0]
    intersection = len(list(set (list1).intersection(list2)))
    union = (len(set (list1)) + len(set (list2))) - intersection
    JaccardSimilarity= float(intersection) / union

    JaccardDissimilarity= 1- JaccardSimilarity


    avgDissimilarity=␣
↪(AgeDissimilarity+SexDissimilarity+MartitalDissimilarity+␣
↪EducationDissimilarity+IncomeDissimilarity+RatingDissimilarity+Persons_in_HouseholdDissimil
↪/ 9
    return (avgDissimilarity.round(3))
```

- Now, we can compute the dissimilarity for each pair of customers.

[11]:
```
dissimilarityFunction(1,3)
```

[11]: 0.502

- If the input is the `id` of the same customer, it returns `0`.

[12]:
```
dissimilarityFunction(1,1)
```

[12]: 0.0

- Now, we create a symmetric matrix to calculate the dissimilarity between the first 5 customers.

[13]:
```
dismatrix = pd.DataFrame(index=np.arange(5), columns=np.arange(5))
dismatrix.index += 1
dismatrix.columns += 1
dismatrix.head()
```

[13]:

	1	2	3	4	5
1	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN

```
3  NaN  NaN  NaN  NaN  NaN
4  NaN  NaN  NaN  NaN  NaN
5  NaN  NaN  NaN  NaN  NaN
```

[14]:
```
#compute disimilarity of 5 customers
for i in range(1, 6):
  for j in range(1, 6):
    if i<=j:
     dismatrix.loc[i,j] = dissimilarityFunction(i,j)
    else:
      pass
```

[15]:
```
dismatrix
```

[15]:
```
     1      2      3      4      5
1  0.0  0.539  0.502  0.167  0.234
2  NaN    0.0  0.391  0.497  0.332
3  NaN    NaN    0.0  0.347  0.558
4  NaN    NaN    NaN    0.0   0.36
5  NaN    NaN    NaN    NaN    0.0
```

0.5.3 Question 3: Nearest Neighbor(NN) search

- To compute the `Nearest Neighbor` algorithm, we create a new function that takes as input the id of a customer and returns the `10` most similar customers. To do that, it computes the similarity with all of the customers according to the previous function(`similarity = 1 - dissimilarity`) and it sorts the result.

[17]:
```
#q3
def NN (a):
  data= pd.DataFrame(index=np.arange(10000), columns=['Customer_ID',␣
 ↪'Similarity_Score'])
  rows= groceries.shape[0] + 1
  for i in range(1, rows):
    if (i== a):
       pass
    else:
       data['Customer_ID'][i] = i
       data['Similarity_Score'][i] = 1- dissimilarityFunction(a,i)
  data=data.sort_values(by=['Similarity_Score'], ascending = False)
  return data.head(10).reset_index().iloc[:, 1:3]
```

[18]:
```
NN(73)
```

[18]:
```
  Customer_ID Similarity_Score
0        1291            0.959
1        1846            0.951
2        1627            0.943
```

```
3         4488             0.929
4         3953             0.924
5         5922             0.923
6         4663             0.922
7         7933             0.918
8         9404             0.917
9         5195             0.914
```

- The previous table shows that the customer with `id = 1291` is more close to customer `73`, next is the customer with `id = 1841`, and so on and so forth.
- And as we can see in the next lines, these 2 customers(with id `73` & `1291`) have very similar characteristics.

[19]:
```
groceries.loc[groceries['Customer_ID'] ==73]
```

[19]:
```
    Customer_ID  Age     Sex Marital_Status  Education  Income  \
72           73   78  female       divorced          3   32000

    Customer_Rating  Persons_in_Household Occupation  \
72                4                     2    retired

                                            Groceries
72  frankfurter,citrus fruit,whole milk,domestic e…
```

[20]:
```
groceries.loc[groceries['Customer_ID'] ==1291]
```

[20]:
```
      Customer_ID  Age     Sex Marital_Status  Education  Income  \
1290         1291   79  female       divorced          3   30000

      Customer_Rating  Persons_in_Household Occupation  \
1290                4                     1    retired

                                    Groceries
1290  pork,beef,whole milk,curd,rolls/buns,pastry
```

- Next, we calculate the 10 nearest neighbors for the customers with the following ids.

[21]:
```
nn = (563, 1603, 2200, 3703, 4263, 5300, 6129, 7800, 8555)
for i in nn:
  print('10 NN for Customer ', i)
  print(NN(i))
  print()
```

```
10 NN for Customer  563
  Customer_ID Similarity_Score
0         3634             0.931
1         6168             0.923
2         6196             0.908
3         2766             0.907
```

```
4        1277            0.899
5         419            0.899
6        4311            0.897
7        8270            0.895
8        7202            0.893
9        7049            0.893

10 NN for Customer  1603
  Customer_ID Similarity_Score
0        7345            0.945
1        4814            0.927
2        7335            0.925
3         568            0.923
4        8959            0.921
5         168            0.916
6        8591            0.909
7        6841            0.898
8        6751            0.891
9        9260             0.89

10 NN for Customer  2200
  Customer_ID Similarity_Score
0         403            0.897
1        7497            0.882
2        6722            0.867
3        8884            0.864
4        3551            0.861
5        5160            0.861
6        5330            0.859
7        4928            0.835
8        6942            0.833
9        2667             0.83

10 NN for Customer  3703
  Customer_ID Similarity_Score
0        9942            0.962
1         374            0.956
2        3352             0.95
3        9419            0.944
4        1604            0.944
5        1837            0.944
6        6587            0.942
7        5853            0.936
8        4838            0.935
9        7847            0.931

10 NN for Customer  4263
  Customer_ID Similarity_Score
```

```
0        5427            0.923
1        2195            0.922
2        9536            0.922
3        3822            0.921
4        5829             0.92
5        4990            0.919
6        9051            0.916
7        1896            0.911
8        2832              0.9
9        2972            0.897

10 NN for Customer  5300
  Customer_ID Similarity_Score
0        2110             0.95
1        8711            0.949
2        8497            0.946
3        3039            0.945
4        8068            0.944
5        3470            0.941
6         326            0.939
7        3533            0.939
8        7542            0.937
9        8982            0.936

10 NN for Customer  6129
  Customer_ID Similarity_Score
0        1082            0.952
1        7870            0.941
2        6387            0.938
3        5301            0.934
4        7563            0.926
5        4933            0.924
6        4856            0.922
7        7837            0.922
8        7557            0.921
9         980            0.917

10 NN for Customer  7800
  Customer_ID Similarity_Score
0        2126            0.914
1         186            0.912
2        2342             0.87
3        9116            0.863
4        7470            0.863
5        8293            0.858
6         673            0.854
7        2506            0.844
8        1847            0.838
```

```
9        8212            0.835

10 NN for Customer  8555
  Customer_ID Similarity_Score
0        1486            0.933
1        6823            0.925
2        8732             0.92
3        3894            0.918
4        7964            0.917
5        3012            0.913
6        7203            0.912
7        4406             0.91
8        6092             0.91
9        2458             0.91
```

0.5.4 Question 4: Customer rating prediction

Part 1:

- For calculating the prediction of the `Custumer Rating` attribute, we had to create, again, another function, similar to the one in the second question (`dissimilarityFunction`), in order to calculate the dissimilarity in all the variables except the one above (`Custumer Rating`).

[22]:
```
#dissimilarityFunction
def dissimilarityFunction_Prediction(a, b):
     # Age
     age1 = groceries['Age'].loc[groceries['Customer_ID'] == a].values[0]
     age2 = b['Age']
     AgeDissimilarity=(abs(age1 - age2)) /(max(groceries['Age']) -␣
 ↪min(groceries['Age']))

     #sex
     sexa = groceries['Sex'].loc[groceries['Customer_ID'] == a].values[0]
     sexb = b['Sex']
     if (sexa==sexb):
        SexSimilarity = 1
     else:
        SexSimilarity = 0

     SexDissimilarity = 1 - SexSimilarity

     #martital
     martitala = groceries['Marital_Status'].loc[groceries['Customer_ID'] == a].
 ↪values[0]
     martitalb = b['Marital_Status']
     if (martitala==martitalb):
        MartitalSimilarity = 1
```

```
    else:
       MartitalSimilarity= 0

    MartitalDissimilarity = 1 - MartitalSimilarity

    # Education
    educationa = groceries['Education'].loc[groceries['Customer_ID'] == a].
↪values[0]
    educationb = b['Education']

    EducationDissimilarity= abs(educationa - educationb) /␣
↪(max(groceries['Education']) - min(groceries['Education']))

    #income
    incomea = groceries['Income'].loc[groceries['Customer_ID'] == a].values[0]
    incomeb = b['Income']
    IncomeDissimilarity=(abs(incomea - incomeb)) /(max(groceries['Income']) -␣
↪min(groceries['Income']))

    #Persons_in_Household
    personsa = groceries['Persons_in_Household'].loc[groceries['Customer_ID']␣
↪== a].values[0]
    personsb = b['Persons_in_Household']
    Persons_in_HouseholdDissimilarity=(abs(personsa - personsb)) /
↪(max(groceries['Persons_in_Household']) -␣
↪min(groceries['Persons_in_Household']))

    #Occupation
    occupationa = groceries['Occupation'].loc[groceries['Customer_ID'] == a].
↪values[0]
    occupationb = b['Occupation']
    if (occupationa==occupationb):
       OccupationSimilarity = 1
    else:
       OccupationSimilarity = 0

    OccupationDissimilarity = 1 - OccupationSimilarity

    #Groceries
    list1= groceries['Groceries'].loc[groceries['Customer_ID'] == a].values[0]
    list2= b['Groceries']
    intersection = len(list(set(list1).intersection(list2)))
    union = (len(set (list1)) + len(set (list2))) - intersection
    JaccardSimilarity= float(intersection) / union

    JaccardDissimilarity= 1- JaccardSimilarity
```

```
    avgDissimilarity=␣
↪(AgeDissimilarity+SexDissimilarity+MartitalDissimilarity+␣
↪EducationDissimilarity+IncomeDissimilarity+Persons_in_HouseholdDissimilarity+OccupationDiss
↪/ 8
    return (avgDissimilarity.round(3))
```

- Later, we made a function called CRP (Customer Rating Prediction), that takes as argument a customer and produces another one with the 10 first more similar to him customers. So, in the end, we have a dataframe showing for that specific customer which other, are less dis-similar from him (more similar).

This function looks like the NN() created above.

[23]:
```
def CRP (a):
  data= pd.DataFrame(index=np.arange(10000), columns=['Customer_ID',␣
↪'Similarity_Score'])
  rows= groceries.shape[0] + 1
  for i in range(1, rows):
      data['Customer_ID'][i] = i
      data['Similarity_Score'][i] = 1 - dissimilarityFunction_Prediction(i,a)
  data=data.dropna()
  data=data.sort_values(by=['Similarity_Score'], ascending = False)
  return data[1:11].reset_index().iloc[:, 1:3]
```

- Next, we designed a function for generating the predicted rating for one customer, based on its 10th nearest neighbors.

[24]:
```
def predict(df):
  cr = list()
  for i in df.iloc[:,0]:
    cr.append(groceries['Customer_Rating'].loc[groceries['Customer_ID'] == i].
↪values[0])

  kv = [key for key, values in mapping1.items() if values == round(sum(cr)/
↪len(cr))]
  return kv
```

- So, in the end we have the prediction of a customer based on the average rating of the 10 most similar customers.

Part 2:

- For calculating the prediction of customer rating, we had to use the weighted average rating of the 10 most similar customers, taking also into account the already having rating. So, we constructed a function called weight in order to predict this rating. The referring function takes a customer from the groceries dataframe, finds its 10 most nearest neighbors from the function created in the previous question (CRP), adds a new column in this dataframe containing the rating of its neighbors (column named `rating`) and then return the predicted value (weighted average).

[25]:
```
def weight(cust):
  df = CRP(cust)
  df['rating'] = ''
  for i in range(len(df.iloc[:,1])):
    df['rating'][i] =  groceries['Customer_Rating'].
↪loc[groceries['Customer_ID'] == df.iloc[i,0]].values[0]

  rating = round(sum(df.iloc[:,1].astype('float64')*df.iloc[:,2]) / sum(df.
↪iloc[:,1]))
  kv = [key for key, values in mapping1.items() if values == rating]
  return kv
```

Evaluation

- For the evaluation of the above classification algorithms we had to use the 50 first records of the `groceries` dataset and predicted the rating for them. Then, we calculated the Mean Prediction Error (MPE) for both prediction methods.

[28]:
```
eval = groceries[:50]
eval.head()
```

[28]:
```
   Customer_ID  Age     Sex Marital_Status  Education  Income  \
0            1   75    male        married          1   20000
1            2   61  female         single          2   28000
2            3   32    male         single          2   34000
3            4   62    male        married          1   31000
4            5   66  female        married          2   19000

   Customer_Rating  Persons_in_Household   Occupation  \
0                4                     3      retired
1                3                     1    housemaid
2                4                     1  blue-collar
3                4                     3  blue-collar
4                3                     3      retired

                                        Groceries
0  citrus fruit,semi-finished bread,margarine,rea…
1                    tropical fruit,yogurt,coffee
2                                      whole milk
3         pip fruit,yogurt,cream cheese,meat spreads
4  other vegetables,whole milk,condensed milk,lon…
```

[27]:
```
mpe1 = list()
mpe2 = list()
for i in range(len(eval.iloc[:,1])):
```

```
  pr = predict(CRP(eval.iloc[i,[0,1,2,3,4,5,7,8,9]]))
  mpe1.append(abs(mapping1[pr[0]]- eval['Customer_Rating'][i]))
  mpe2.append(abs(mapping1[weight(eval.iloc[i,[0,1,2,3,4,5,7,8,9]])[0]]-␣
 ↪eval['Customer_Rating'][i]))

mpe_fin1 = sum(mpe1)/len(eval.iloc[:,1])
mpe_fin2 = sum(mpe2)/len(eval.iloc[:,1])
print('Mean Prediction Error for the 1st Method:', mpe_fin1)
print('Mean Prediction Error for the 2nd Method:', mpe_fin2)
```

```
Mean Prediction Error for the 1st Method: 0.7
Mean Prediction Error for the 2nd Method: 0.72
```

- As we can observe from the output of the above script, the Mean Prediction Error for both the two methods is nearly the same. In the first case is nearly `0,7` and in the second is `0,72`. This error shows, on average, how much deviation the expected values of a customer rating have from their actual values. The bigger it is, the worse. So, the values found earlier are good indications of well predicted methods. Because of the fact that if we round these predictors will not exceed the `1` unit, we can also conclude that if the actual rating of a customer to a supermarket is `good` then the predictors will give us either `good`, `fair` or `very good`.

Conclusions

The above script is a useful tool for one company, such as supermarket, in order to predict the rating of its customers, based on their demographic characteristics. If in this company a new client is going to be added, then based on the other customers and how similar they are with him, we can predict his/her rating with really big precision. This tool adds big value to a company, because by exploiting it, it could take various decisions concerning the marketing field, such as more directed advertisement.

Finally, it would be a good idea if the company evolves this script by adding more methods like clustering of the customers. But, before doing this, the company should consider its resources, because each computation in order to run correctly needs a lot of time.