Phase_2_CF

November 30, 2019

0.0.1 Type III: Collaborative Filtering based Recommendation System

It works on the basic idea that people with similar characteristics have similar interests in movies.

There are 2 types of Collaborative Filtering Techniques:

> *User-Based*: Here we assume that similar people have similar liking. If user A and user B have watched a lot of similar movies and have rated them almost equally, then it can be inferred that their interests match. Thus we can recommend movies that A have liked to B.
>
> *Item-Based*: Here we take into consideration the content of movies as well and not only user similarity. Here we find those movies which the user would enjoy based on his previously rated movies.

User Based Collaborative Filtering is a very common and highly used approach for movielens dataset because of the fact that:

- It is easy to implement.
- It is context independent.
- It gives better results that content based recommendation system.

But this approach is not the best. Recommending movies without taking into consideration the content of the movie would never give best results.

Thus we tried to implement an **item based recommendation system** with our movies dataset.

Firstly, we start of by reading the data. To find similarity between items, we are using 5 different types of feature combinations:

1. Genre, Crew
2. Cast, Keywords
3. Genre, Crew, Production Countries
4. Genre, Production Countries, Keywords
5. Genre, Crew, Production Countries, Keywords

The above combinations were made on our basic understanding of what all features influence our liking for a movie.

The data is combined from multiple files: keywords.csv, credits.csv, movies_metadata.csv, ratings.csv

Mount & Imports

[0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[0]:
```
cd /content/drive/My\ Drive/DataScience\ Project/the-movies-dataset
```

```
/content/drive/My Drive/DataScience Project/the-movies-dataset
```

[0]:
```
ls
```

```
 BestFeatureVector.csv    Errors1.csv    links_small.csv
 cosSimScore.csv          Errors-S.csv   movies_metadata.csv
 credits.csv              keywords.csv   ratings.csv
'Customer Feedback.csv'   links.csv      ratings_small.csv
```

[0]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import MultiLabelBinarizer
from nltk.corpus import stopwords
from ast import literal_eval
import pandas as pd, numpy as np,matplotlib.pyplot as plt
import nltk, re, string
from prettytable import PrettyTable
from prettytable import from_csv
nltk.download('stopwords')
stopWord = set(stopwords.words('english'))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

Functions

[0]:
```
def removeStopWord(tokens): #Function removing stop words and punctuation
    global stopWord
    low = tokens.lower()
    cl = re.compile('_') #remove punctuation
    loweredTokens = re.split(';|_|!|#|@|$|%|\^|&|\*|\(|\)|:|\'|\"| |\.|\,|',low)
    fr = []
    for i in loweredTokens:
        if(i not in stopWord):
            fr.append(i)
    return (' ').join(fr)
```

[0]:

```python
def binarise(y_train): #Function to create a binary vector for features
  mlb = MultiLabelBinarizer()
  y_Bin = mlb.fit_transform(y_train)
  clsNames = mlb.classes_
  return y_Bin, clsNames
```

[0]:

```python
def vectorForm(df,col): #Creates feature vector
    temp = [[] for i in range(len(df))]
    colNames = []
    for i in col:
        val,colName = binarise(df[i])
        temp = np.hstack((temp,val))
        colNames.extend(colName)
    vectDf = pd.DataFrame(temp,columns=colNames)
    finlVec = pd.concat([vectDf,df.rating],axis=1)
    return finlVec
```

[0]:

```python
def recommend(test,r,sim,mu,title,n): #Recommends top n movies
    predRating = dict()
    for i in test.index:
        sum1 = 0
        sum2 = 0
        for j in test.index:
            sum1 += (sim[i][j]*(r[j]-mu))
            sum2 += sim[i][j]
        predRating[i] = round(mu + (sum1/sum2),2)
    sortedScore = sorted(predRating.items(), key=lambda x: x[1],reverse=True)␣
 ↪#Sorting in decreasing order on the basis of similarity score
    topScores = sortedScore[1:n+1]
    movies = [i[0] for i in topScores]
    return list(title[movies])
```

Reading Data and preprocessing

[0]:

```python
keyDf = pd.read_csv('keywords.csv')
credDf = pd.read_csv('credits.csv')
metaDf = pd.read_csv('movies_metadata.csv')
ratDf = pd.read_csv('ratings.csv')
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718:
DtypeWarning: Columns (10) have mixed types. Specify dtype option on import or
set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

[0]:

```python
print(keyDf.shape,credDf.shape,metaDf.shape,ratDf.shape)
```

```
(46419, 2) (45476, 3) (45466, 24) (26024289, 4)
```

```
[0]: ratDf.head()
```

```
[0]:    userId  movieId  rating   timestamp
     0       1      110     1.0  1425941529
     1       1      147     4.5  1425942435
     2       1      858     5.0  1425941523
     3       1     1221     5.0  1425941546
     4       1     1246     5.0  1425941556
```

```
[0]: ratDf.rename(columns = {"movieId": "id"},inplace=True) #Renaming movieId col to␣
      ↪id for merging
```

```
[0]: ratDf.columns
```

```
[0]: Index(['userId', 'id', 'rating', 'timestamp'], dtype='object')
```

```
[0]: metaDf.head(2)
```

```
[0]:    adult  … vote_count
     0  False  …     5415.0
     1  False  …     2413.0

     [2 rows x 24 columns]
```

```
[0]: keyDf.head()
```

```
[0]:       id                                           keywords
     0    862  [{'id': 931, 'name': 'jealousy'}, {'id': 4290,…
     1   8844  [{'id': 10090, 'name': 'board game'}, {'id': 1…
     2  15602  [{'id': 1495, 'name': 'fishing'}, {'id': 12392…
     3  31357  [{'id': 818, 'name': 'based on novel'}, {'id':…
     4  11862  [{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n…
```

```
[0]: credDf.head()
```

```
[0]:                                                cast  …     id
     0  [{'cast_id': 14, 'character': 'Woody (voice)',…  …    862
     1  [{'cast_id': 1, 'character': 'Alan Parrish', '…  …   8844
     2  [{'cast_id': 2, 'character': 'Max Goldman', 'c…  …  15602
     3  [{'cast_id': 1, 'character': "Savannah 'Vannah…  …  31357
     4  [{'cast_id': 1, 'character': 'George Banks', '…  …  11862

     [5 rows x 3 columns]
```

Info about DF

[0]:

```
print(keyDf.info())
print()
print(credDf.info())
print()
print(metaDf.info())
print()
print(ratDf.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46419 entries, 0 to 46418
Data columns (total 2 columns):
id          46419 non-null int64
keywords    46419 non-null object
dtypes: int64(1), object(1)
memory usage: 725.4+ KB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45476 entries, 0 to 45475
Data columns (total 3 columns):
cast    45476 non-null object
crew    45476 non-null object
id      45476 non-null int64
dtypes: int64(1), object(2)
memory usage: 1.0+ MB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
adult                    45466 non-null object
belongs_to_collection    4494 non-null object
budget                   45466 non-null object
genres                   45466 non-null object
homepage                 7782 non-null object
id                       45466 non-null object
imdb_id                  45449 non-null object
original_language        45455 non-null object
original_title           45466 non-null object
overview                 44512 non-null object
popularity               45461 non-null object
poster_path              45080 non-null object
production_companies     45463 non-null object
production_countries     45463 non-null object
release_date             45379 non-null object
revenue                  45460 non-null float64
runtime                  45203 non-null float64
```

```
spoken_languages         45460 non-null object
status                   45379 non-null object
tagline                  20412 non-null object
title                    45460 non-null object
video                    45460 non-null object
vote_average             45460 non-null float64
vote_count               45460 non-null float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26024289 entries, 0 to 26024288
Data columns (total 4 columns):
userId       int64
id           int64
rating       float64
timestamp    int64
dtypes: float64(1), int64(3)
memory usage: 794.2 MB
None
```

Converting int to str

[0]:
```python
keyDf.id = keyDf.id.apply(str)
credDf.id = credDf.id.apply(str)
ratDf.id = ratDf.id.apply(str)
```

Sampling of data

[0]:
```python
maxUserRat = list(ratDf.userId[ratDf.userId == ratDf.userId.value_counts().index[0]].index) #Picking users with maximum ratings count i.e. 45811
sampleRatDf = ratDf.iloc[maxUserRat,:] #Sampling maximum users data based on ratings
sampleRatDf.drop(['userId','timestamp'],axis=1,inplace=True) #Dropping off timestamp & userId
sampleRatDf.index = range(len(sampleRatDf)) #Resetting indexes
sampleRatDf.head()
```

```
/usr/local/lib/python3.6/dist-packages/pandas/core/frame.py:4117:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  errors=errors,
[0]:    id  rating
     0   1     4.0
     1   2     3.0
     2   6     4.0
     3   9     2.5
     4  10     3.0
```

Merging on movieid column

```
[0]: merKeyDf = metaDf.merge(keyDf,on='id')
     merCredDf = merKeyDf.merge(credDf,on='id')
     mergedDf = merCredDf.merge(sampleRatDf,on='id')
```

```
[0]: mergedDf.shape
```

```
[0]: (3634, 28)
```

```
[0]: finalDf =␣
      ↪mergedDf[['id','title','genres','keywords','cast','crew','production_countries','rating']]
     finalDf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3634 entries, 0 to 3633
Data columns (total 8 columns):
id                      3634 non-null object
title                   3634 non-null object
genres                  3634 non-null object
keywords                3634 non-null object
cast                    3634 non-null object
crew                    3634 non-null object
production_countries    3634 non-null object
rating                  3634 non-null float64
dtypes: float64(1), object(7)
memory usage: 255.5+ KB
```

Preprocessing (Converting structures to list & removing stop words or punctutation)

```
[0]: finalDf['genres'] = finalDf['genres'].fillna('[]').apply(literal_eval).
      ↪apply(lambda x: [removeStopWord(i['name']) for i in x] if isinstance(x,␣
      ↪list) else [])
     finalDf['cast'] = finalDf['cast'].fillna('[]').apply(literal_eval).apply(lambda␣
      ↪x: [removeStopWord(i['name']) for i in x] if isinstance(x, list) else [])
     finalDf['crew'] = finalDf['crew'].fillna('[]').apply(literal_eval).apply(lambda␣
      ↪x: [removeStopWord(i['name']) for i in x] if isinstance(x, list) else [])
     finalDf['production_countries'] = finalDf['production_countries'].fillna('[]').
      ↪apply(literal_eval).apply(lambda x: [removeStopWord(i['name']) for i in x]␣
      ↪if isinstance(x, list) else [])
```

```
finalDf['keywords'] = finalDf['keywords'].fillna('[]').apply(literal_eval).
 ↪apply(lambda x: [removeStopWord(i['name']) for i in x] if isinstance(x,␣
 ↪list) else [])
```

```
/usr/lib/python3.6/re.py:212: FutureWarning: split() requires a non-empty
pattern match.
  return _compile(pattern, flags).split(string, maxsplit)
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  This is separate from the ipykernel package so we can avoid doing imports
until
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:5:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  """
```

[0]:
```
finalDf.head()
```

[0]:
```
      id  … rating
0  8844  …    4.0
1   949  …    4.0
2  1408  …    3.5
3   524  …    4.0
4  8012  …    3.5

[5 rows x 8 columns]
```

Building Recommendation System The recommendation system is build by following following steps:

Following set of feature combinations is used: 1. Genre, Crew 2. Crew, Keywords 3. Genre ,Crew ,Production Countries 4. Genre, Production Countries, Keywords 5. Genre, Crew, Production Countries, Keywords

Below Steps are followed for all combinations of features: 1. Given dataset for a specific user is split into train - test ratio. 2. Using the training data, where we have user rating corresponding to each movie vector, we train the regressor to predict user's ratings on movies from the test dataset. 3. Now we have predicted user rating for movie i and average rating of movie by all the users. 4. Using RMSE and MAE, we find the est set of features which is able to generalise the user's interests.

Using the best set of features, we find cosine similarity between movies features. Using the below mentioned formula, we find rating of movie for a user.

Using predicted rating for each movie, we sort the best rated movies for user 'u' and recommend.

[0]:
```
'''Dictionary for few combination of features'''

feat = {0:['genres','crew'],
        1:['Crew','keywords'],
        2:['genres','crew','production_countries'],
        3:['genres','production_countries','keywords'],
        4:['genres','crew','production_countries','keywords']
        }
```

[0]:
```
'''Running for all combination and calculating MAE and RMSE'''
testrmse = []
testmae = []
trainrmse = []
trainmae = []
j=0
for i in feat.values():
    j+=1
    # print('\nFor: ',i)
    vector = vectorForm(finalDf,i)
    vector = vector.values
    X = vector[:,:-1]
    y = vector[:,-1]
    y = y.astype(float)
    print(X.shape)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,shuffle=False)
    X_tr, X_ts, y_tr, y_ts = train_test_split(X_train, y_train, test_size=0.
 ↪2,shuffle=False)
    clftst = KNeighborsRegressor(n_neighbors=5)
    clftrn = KNeighborsRegressor(n_neighbors=5)
    clftst.fit(X_train, y_train)
    clftrn.fit(X_tr,y_tr)
    predTest = clftst.predict(X_test)
    predTrain = clftrn.predict(X_ts)
    trm = mean_absolute_error(y_ts,predTrain)
    trr = mean_squared_error(y_ts,predTrain)
    tsm = mean_absolute_error(y_test,predTest)
    tsr = mean_squared_error(y_test,predTest)
    testrmse.append(np.sqrt(tsr))
    testmae.append(tsm)
    trainrmse.append(np.sqrt(trr))
    trainmae.append(trm)
    x.add_row([str(j), i, X.shape, trm, np.sqrt(trr),tsm, np.sqrt(tsr) ])
```

[0]:

```
scoreDf.to_csv('Errors.csv',index=None) #Dumping errors in csv
```

[0]:

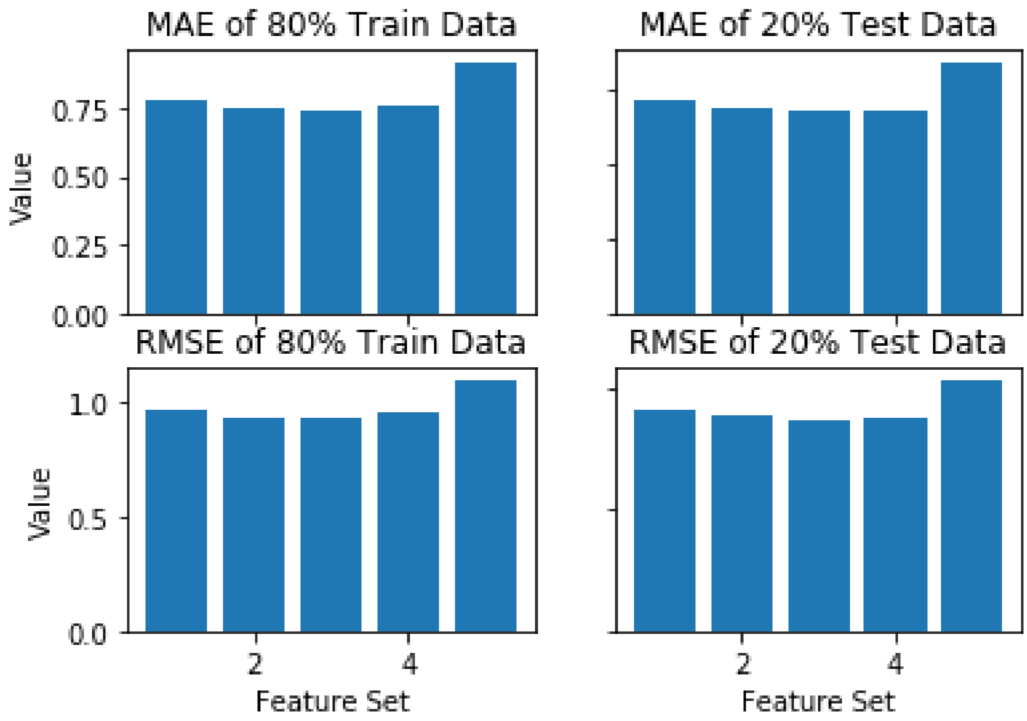
```
x = [1,2,3,4,5]
ylbl = [0.0,0.2,0.4,0.6,0.8,1.0]
error = pd.read_csv('Errors1.csv')

fig, axs = plt.subplots(2, 2)
axs[0, 0].bar(x,error['Train MAE'])
axs[0, 0].set_title('MAE of 80% Train Data')
axs[0, 1].bar(x,error['Test MAE'])
axs[0, 1].set_title('MAE of 20% Test Data')
axs[1, 0].bar(x,error['Train RMSE'])
axs[1, 0].set_title('RMSE of 80% Train Data')
axs[1, 1].bar(x,error['Test RMSE'])
axs[1, 1].set_title('RMSE of 20% Test Data')

for ax in axs.flat:
    ax.set(xlabel='Feature Set', ylabel='Value')

for ax in axs.flat:
    ax.label_outer()
plt.show()


with open("Errors-S.csv", "r") as fp:
    x = from_csv(fp)
print(x)
```



```
+---------------------------------------------+--------------+--------------+---
-----------+--------------+
|                   Features                  |   Test MAE   |  Test RMSE   |  
Train MAE    |  Train RMSE  |
+---------------------------------------------+--------------+--------------+---
-----------+--------------+
|                  Genre, Crew                | 0.6829436039 | 0.8827476799 | 
0.7599656357 | 0.9556038375 |
|                 Crew, Keywords              | 0.8418156809 | 1.036022444  | 
0.9125429553 | 1.097067318  |
|      Genre ,Crew ,Production Countries      | 0.7138927098 | 0.9162599656 | 
0.776975945  | 0.9657538053 |
|    Genre, Production Countries, Keywords    | 0.6777166437 | 0.8672158173 | 
0.7374570447 | 0.9352398299 |
| Genre, Crew, Production Countries, Keywords | 0.6905089409 | 0.8905587631 | 
0.7451890034 | 0.9341828485 |
+----------------------------------------------+--------------+--------------+---
-----------+--------------+
```

RMSE value close to 0 indicates that the regression model is able to generalise well. We take the set of attributes for which we get the lowest RMSE and MAE Errors. This our case, the required set of attriutes is: Genre, Production Countries, Keywords

We can also see that Test RMSE and Train RMSE are almost equal. Thus we can conclude that our model doesn't overfit/ underfit on the data.

Choosing Best Set of features according to minimum error

[0]:
```
mmin = 1
rmin = 1
for i in range(len(testrmse)):
    if(mmin > testmae[i] and rmin > testrmse[i]):
        mmin = testmae[i]
        rmin = testrmse[i]
```

[0]:
```
print(mmin,rmin)
```

```
0.6777166437414031 0.8672158172725218
```

[0]:
```
bestFeat = feat[testmae.index(mmin)]
print(bestFeat)
```

```
['genres', 'production_countries', 'keywords']
```

[0]:
```
bestFeatVector = vectorForm(finalDf,bestFeat)
bestFeatVector.to_csv('BestFeatureVector.csv',index=None) #Dumpping best␣
 ↪feature vector based on min MAE and RMSE to csv
```

Cosine Similarity

[0]:
```
bfv = pd.read_csv('BestFeatureVector.csv') #Dumpping best feature vector in csv
simMatrix = cosine_similarity(bfv.values,bfv.values)
pd.DataFrame(simMatrix).to_csv('cosSimScore.csv',index=None) #Dumpping␣
 ↪similarity matrix in csv file
simMatrix = pd.read_csv('cosSimScore.csv')
# simMatrix.shape
sim = simMatrix.values
mu = round(bfv.rating.mean(),2) #Mean rating for user who has done max ratings
X = bfv.iloc[:,bfv.columns!='rating']
y = bfv.rating
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪2,shuffle=False)
```

Finding Recommendations

[0]:
```
recommend(X_test,y_test,sim,mu,finalDf.title,10)
```

```
[0]: ['Due cuori, una cappella',
      'Samay: When Time Strikes',
      'Virasat',
      'The Last Relic',
      'The Colour Out of Space',
      'Shocking Asia',
      'Koyla',
      'The Incredible Kung Fu Master',
      'The Mystery of Chess Boxing',
      'Afterparty']
```