nmf_mov_rating

May 28, 2024

[6]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import time
import sklearn
from scipy.sparse import coo_matrix, csr_matrix
from scipy.spatial.distance import jaccard, cosine
from sklearn.metrics import pairwise_distances
from sklearn.decomposition import NMF
from pytest import approx
```

[16]:
```
path = 'Downloads/unsupervised/Files/Module3/'
MV_users = pd.read_csv(path+'data/users.csv')
MV_movies = pd.read_csv(path+'data/movies.csv')
train = pd.read_csv(path+'data/train.csv')
test = pd.read_csv(path+'data/test.csv')
test.info(),train.info()
```

[16]:
```
array([4, 5, 3, 2, 1])
```

[8]:
```
from collections import namedtuple
Data = namedtuple('Data', ['users','movies','train','test'])
data = Data(MV_users, MV_movies, train, test)
```

PART 1

We will break the Movie rating matrix, self.Mr into components(ets say equal to number of genres) in the W matrix. Then we will multiply W and H back to get an esitmate of movie ratings matrix again, hoping that this might have missing ratings accounted for in a better way. Then we will use this W*H product to predict the test ratings. This gives us a low value of RMSE.

[46]:
```
class RecSys():
    def __init__(self,data):
        self.data=data
        self.allusers = list(self.data.users['uID'])
        self.allmovies = list(self.data.movies['mID'])
        self.genres = list(self.data.movies.columns.drop(['mID', 'title',␣
 ↪'year']))
```

```
        self.mid2idx = dict(zip(self.data.movies.mID,list(range(len(self.data.
↪movies)))))
        self.uid2idx = dict(zip(self.data.users.uID,list(range(len(self.data.
↪users)))))
        self.Mr=self.rating_matrix()
        self.Mm=None
        self.sim=np.zeros((len(self.allmovies),len(self.allmovies)))

    def rating_matrix(self):
        """
        Convert the rating matrix to numpy array of shape (#allusers,#allmovies)
        """
        ind_movie = [self.mid2idx[x] for x in self.data.train.mID]
        ind_user = [self.uid2idx[x] for x in self.data.train.uID]
        rating_train = list(self.data.train.rating)

        return  np.array(coo_matrix((rating_train, (ind_user, ind_movie)),␣
↪shape=(len(self.allusers), len(self.allmovies))).toarray())



    def Nmf(self):
        C = (self.Mr)

        X = NMF(n_components=len(self.genres),solver='mu',
                                    beta_loss="frobenius",alpha_W=0.
↪00005,alpha_H=0.00005,l1_ratio=0.5,).fit(C)
        W = X.transform(C)
        H = X.components_
        return W@H

    def predict(self):
        """
        Predict ratings in the test data. Returns predicted rating in a numpy␣
↪array of size (# of rows in testdata,)
        """
        self.Mr = self.Nmf()

        return np.array([self.Mr[self.uid2idx[uid],self.mid2idx[mid]] for␣
↪(uid,mid) in zip(self.data.test.uID,self.data.test.mID)])



    def rmse(self,yp):
        yp[np.isnan(yp)]=3 #In case there is nan values in prediction, it will␣
↪impute to 3.
        yt=np.array(self.data.test.rating)
```

```
        return np.sqrt(((yt-yp)**2).mean())
```

[47]:
```
sample_cb = RecSys(data)
sample_yp = sample_cb.predict()
sample_rmse = sample_cb.rmse(sample_yp)

print(sample_rmse)
```

```
2.868054687603766
```

PART 2

This method failed because it is a bad method. Now in part 2: We factorized the matrix using the number of genres as latent dimension. Then we constructed the similarity matrix to get the predictions. This method gives us very good results and an RMSE of 0.97. First the matrix is broken by NMF into a movie matrix of dimension :(len(movies),len(genres)). First this matrix is converted to boolean matrix. Values in this matrix lower than mean are made zero and more than mean made 1. Now, this matrix is converted to similarity matrix of shape (len(movies),len(movies)) by using jaccard similarity as in week 3 assignment. Then everything works the same way as week 3 assignment, i.e., predict_from_sim function is applied on every entry in test data. Finally, RMSE is calculated.

[48]:
```
class RecSys():
    def __init__(self,data):
        self.data=data
        self.allusers = list(self.data.users['uID'])
        self.allmovies = list(self.data.movies['mID'])
        self.genres = list(self.data.movies.columns.drop(['mID', 'title',␣
 ↪'year']))
        self.mid2idx = dict(zip(self.data.movies.mID,list(range(len(self.data.
 ↪movies)))))
        self.uid2idx = dict(zip(self.data.users.uID,list(range(len(self.data.
 ↪users)))))
        self.Mr=self.rating_matrix()
        self.Mm=None
        self.sim=np.zeros((len(self.allmovies),len(self.allmovies)))

    def rating_matrix(self):
        """
        Convert the rating matrix to numpy array of shape (#allusers,#allmovies)
        """
        ind_movie = [self.mid2idx[x] for x in self.data.train.mID]
        ind_user = [self.uid2idx[x] for x in self.data.train.uID]
        rating_train = list(self.data.train.rating)
```

```
        return np.array(coo_matrix((rating_train, (ind_user, ind_movie)),␣
↪shape=(len(self.allusers), len(self.allmovies))).toarray())


    def predict_from_sim(self,uid,mid):
        """
        Predict a user rating on a movie given userID and movieID
        """
        r = self.Mr[self.uid2idx[uid]]
        s = self.sim[self.mid2idx[mid]]
        idx = np.where(r>0)
        return np.dot(r,s)/(s[idx].sum()+0.01)
    def predict(self):
        """
        Predict ratings in the test data. Returns predicted rating in a numpy␣
↪array of size (# of rows in testdata,)
        """
        return np.array([self.predict_from_sim(uid,mid) for (uid,mid) in␣
↪zip(self.data.test.uID,self.data.test.mID)])


    def rmse(self,yp):
        yp[np.isnan(yp)]=3 #In case there is nan values in prediction, it will␣
↪impute to 3.
        yt=np.array(self.data.test.rating)
        return np.sqrt(((yt-yp)**2).mean())

    def Nmf(self):
        C = csr_matrix(self.Mr.T)
        X = NMF(n_components=len(self.genres),solver='mu',
                                    beta_loss="frobenius",alpha_W=0.
↪00005,alpha_H=0.00005,l1_ratio=0.5,).fit(C)
        W = X.transform(C)
        return W
    def calc_item_item_similarity(self):
        """
        Create item-item similarity using Jaccard similarity
        """
        ##CONVERT TO BOOLEAN ARRAY By comparing mean value and then use␣
↪jaccardian
        arr_MM = self.Nmf()
        arr_bool = np.array([arr_MM[i]>arr_MM[i].mean() for i in␣
↪range(len(arr_MM))])
        self.sim = 1 - pairwise_distances(arr_bool, metric="jaccard")
```

[50]:
```
sample_cb = RecSys(data)
sample_cb.calc_item_item_similarity()
sample_yp = sample_cb.predict()
sample_rmse = sample_cb.rmse(sample_yp)

print(sample_rmse)
```

```
0.9706142010871062
```

We factorized the matrix using the number of genres as latent dimension. Then we constructed the similarity matrix to get the predictions. This method gives us very good results and an RMSE of 0.97. First the matrix is broken by NMF into a movie matrix of dimension :(len(movies),len(genres)). Then this matrix is converted to boolean matrix. Values in this matrix lower than mean are made zero and more than mean made 1. Now, this matrix is converted to similarity matrix of shape (len(movies),len(movies)) by using jaccard similarity as in week 3 assignment. Then everything works the same way as week 3 assignment, i.e., predict_from_sim function is applied on every entry in test data. Finally, RMSE is calculated. The result in the RMSE category is better with Nmf but the individual predictions are not so good. RMSE means the root of mean of the total sample error. The RMSE is better than those of baseline methods as clearly can be seen.

[ ]:
```

```