Movie Recommender System using Non-negative Matrix Factorization.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.sparse import coo_matrix, csr_matrix
from pytest import approx
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn import decomposition
```

First we load the data used in week 3 movie recommendation and create the same Data object as in week 3.

In [2]:
```
MV_users = pd.read_csv('data/users.csv')
MV_movies = pd.read_csv('data/movies.csv')
train = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')
```

In [3]:
```
from collections import namedtuple
Data = namedtuple('Data', ['users','movies','train','test'])
data = Data(MV_users, MV_movies, train, test)
```

1. Same class initialization as in week 3 homework to create the user-movie matrix and then a function that does NMF from the sklearn library on such matrix and use the results to predict the missing values. For the NMF I use a small regularization value, and start by looking at 5 components. To get the predicted values, the matrices H and W that result from NMF are multiplied to a get a matrix that gives the predictions.

In [15]:
```
class RecSys():
    def __init__(self, data, nc):
        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.predicted = self.NMF_fact_1(nc,0.001)
        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(train.rating)
        return np.array(coo_matrix((rating_train, (ind_user, ind_movie)), shape=(len(sel

    def predict_from_NMF(self,uid,mid):
        """
        Use matrix factorization result to get the prediction.
```

```
        Search the predicted value in the predicted matrix.
        """
        return self.predicted[self.uid2idx[uid],self.mid2idx[mid]]


    def predict(self):
        """
        Predict ratings in the test data. Returns predicted rating in a numpy array of s
        """
        # your code here
        ind_user =  self.data.test.uID
        ind_movie = self.data.test.mID
        rat_test = np.zeros(len(ind_user),)
        for i in range(0,len(ind_movie)):
            rat_test[i] = self.predict_from_NMF(ind_user[i],ind_movie[i])
        return rat_test

        pass

    def NMF_fact_1(self,n_comp,alpha):
        """
        Uses NMF factorization once to create a predicted matrix. By factoriz the matrix
        in two matrices with n_componets. And then multiplying them back together to get
        the values of the missing ratings.
        """
        self.nmf_mod = decomposition.NMF(n_components = n_comp, alpha = alpha, max_iter
        W = self.nmf_mod.fit_transform(self.Mr)
        H = self.nmf_mod.components_
        return W.dot(H).clip(1,5)


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


    def update_NMF(self,ite_num):
        W = self.nmf_mod.fit_transform(self.Mr)
        H = self.nmf_mod.components_
        for i in range(0,ite_num):
            new = W.dot(H)
            aux = np.where(self.Mr==0, new, self.Mr)
            W = self.nmf_mod.fit_transform(aux)
            H = self.nmf_mod.components_

        return(W.dot(H).clip(1,5))
```

I start by setting the number of components of the NMF algorithm to 5 and check the rmse value of the test data.

In [16]:
```
rs = RecSys(data,nc = 5)
yp = rs.predict()
print(rs.rmse(yp))
```

```
2.6086382547148195
```

1. - The rmse result is 2.61. Thats a really high value for the rmse, twice the value of setting all the values to 3. For any data between 1 and 5 if you set the values to 3 the biggest error you can get is 2 in case the result would be 5. Here the error is even more than half the range of the data. As rmse is just the square root of the sum of the square of all the diferences between the real value and the predicted value given by the matrix created by multiplying W and H over the total number of predictions tried, so Is a way of getting the average error of the recommendation. The largest average error possible for this data is 4, as the data goes from 1 to 5. So having and average error of 2.6 is worst than random guessing.

To try to improve the error we do a search over the number of components to see which one gives a lower rmse.

In [ ]:
```
rmse = [rs.rmse(yp)]
for n_c in range (6,30):
    rs.predicted = rs.NMF_fact_1(n_c, alpha = 0.001)
    yp = rs.predict()
    rmse.append(rs.rmse(yp))
rmse
```

Even whe changing the loss function, and the regularization parameter the rmse values keep being higher than the results we found in homework 3. The best results were around 2.534 for 18 and 22 number of components, thats still a terrible result. As we can see from the warnings even when set at 450 the number of iterations aren't enough to get to the tolerance level of the algorithm. The matrix we are working with to make the recommendations is quite big and sparse so the results aren't good. As the resulting from multiplying HW we are getting a matrix with a lot of small values that become one when we restrict the values to be between 1 and 5. A way to improve the poor results due to the sparsity we can create a new matrix to fit that would be created by combining the original matrix rs.MR that we want to fit a fill the zeros of such matrix with the results of the matrix thats a result o H times W. And repeat it with the new factorization.
https://archive.siam.org/meetings/sdm06/proceedings/059zhangs2.pdf

We start by setting the number of components to 18. And then we do the recursive method explained above to improve the prediction results of the NMF.

In [20]:
```
rs = RecSys(data,nc = 18)
yp = rs.predict()
print(rs.rmse(yp))
rs.predicted = rs.update_NMF(50)
yp = rs.predict()
print(rs.rmse(yp))
```

```
2.5346597390051997
1.0443547984464792
```

After 50 iterations the rmse improved from 2.53 to 1.04 which is a great improve but still worst than setting every user rating to its average value. So I wouldnt recommend NMF for recommendation systems where the data is sparse as all the collaborative methods used in homework 3 have better results.

In [ ]: