# Lab Machine Learning 10

## Souaybou Bagayoko

### Semester 2, Nr: 303189

¶

## Exercise 1: Recommender Dataset

In [102]:
```
#import the libraries

import numpy as np
import pandas as pd
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits import mplot3d
import seaborn as sns
from sklearn.preprocessing import  Normalizer
from sklearn.metrics import mean_squared_error
from  scipy.sparse import csr_matrix
from IPython.core.debugger import set_trace
from collections import Counter, defaultdict
import matplotlib.pyplot as plt
%matplotlib inline
```

In [129]:
```
#reading the data
ratings = pd.read_csv('ml-100k/u.data', delim_whitespace=True, header=
None, \
                   names=['user_id', 'item_id', 'ratings','timestamp']
)
```

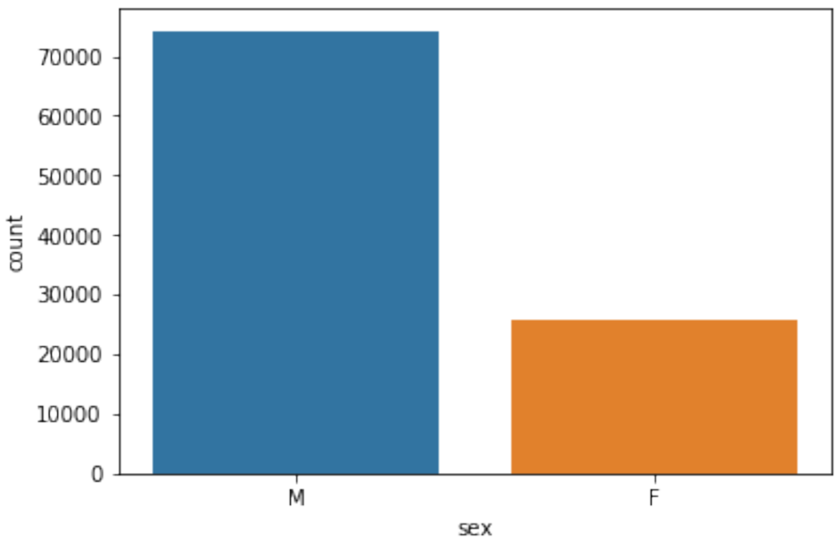
In [131]:
```
user_info = pd.read_csv('ml-100k/u.user', sep='|',\
                   names=['user_id','age', 'sex', 'occupation','zipcod
e'])
```

In [132]:
```
data = pd.merge(ratings, user_info, on='user_id')
```

## some useing ploting

In [133]:
```
sns.countplot(x='sex', data=data)
```

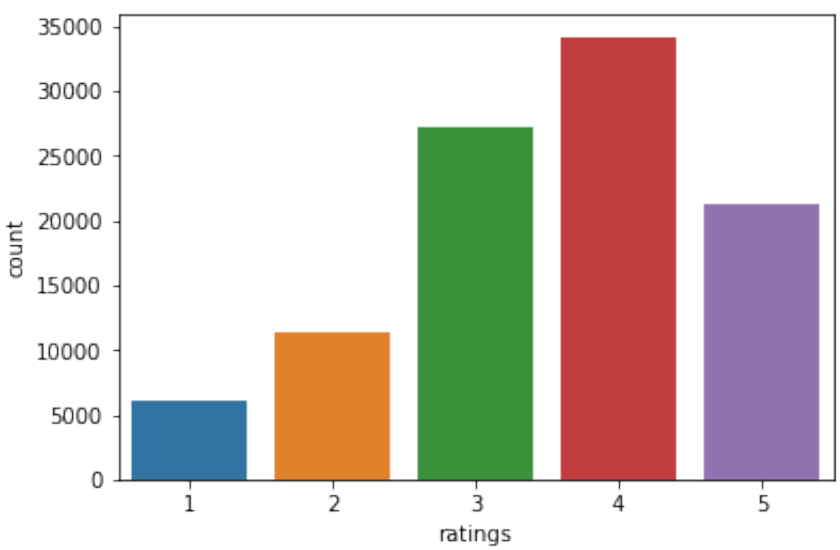
Out[133]: <matplotlib.axes._subplots.AxesSubplot at 0x1a419a36d8>




## femal are more like to rate the movies than male

In [134]:
```
sns.countplot(x='ratings', data=data)
```

Out[134]: <matplotlib.axes._subplots.AxesSubplot at 0x1a419c87f0>

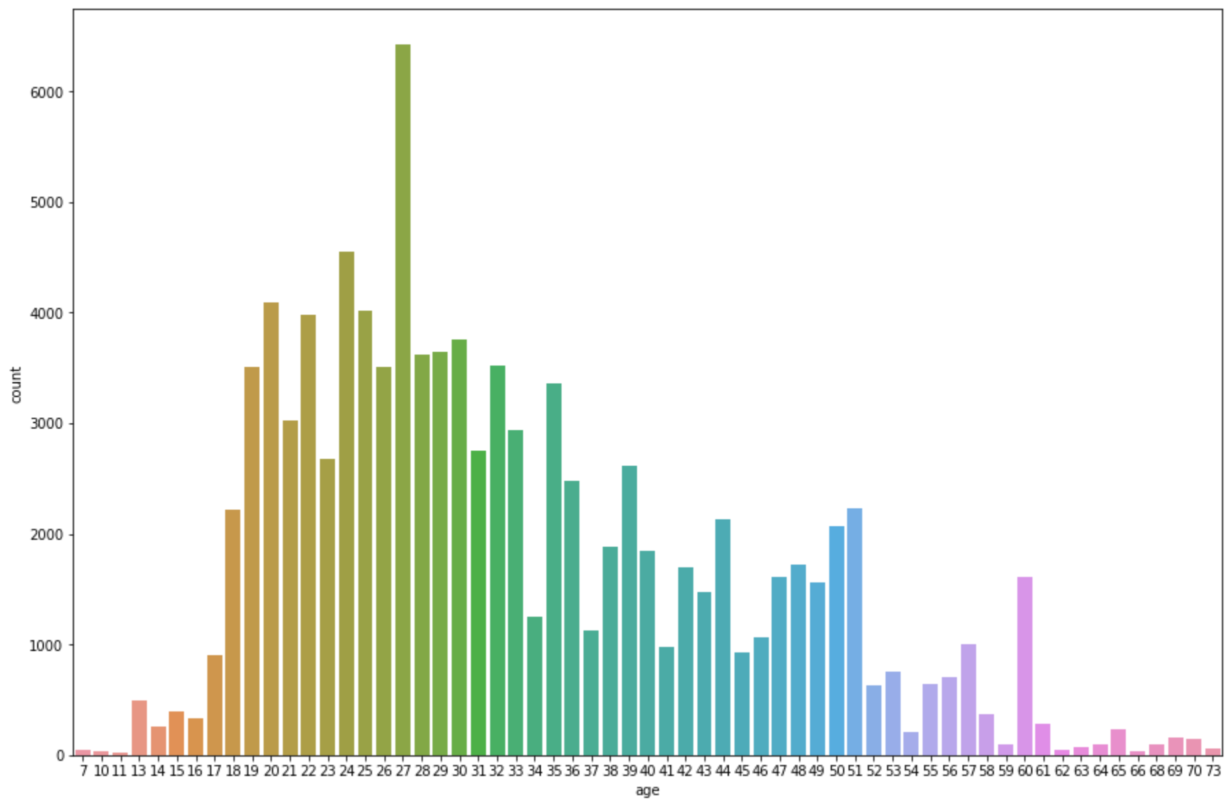


## We Notice some interesting behavior:

```
the user are tend to give 4,3,5 as grating, they are more enthousiathic in grading
```

In [135]:
```
fig_dims = (15, 10)
fig, ax = plt.subplots(figsize=fig_dims)
sns.countplot(x = "age", ax=ax, data=data)
```

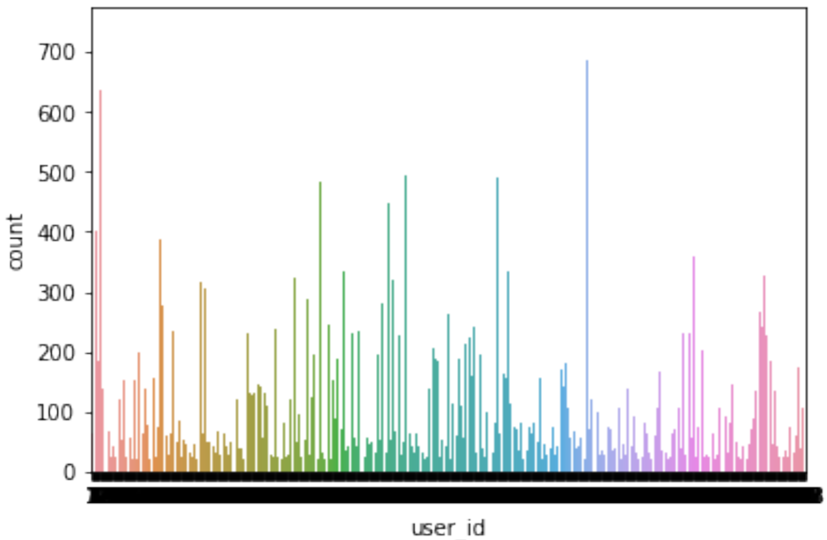
Out[135]: <matplotlib.axes._subplots.AxesSubplot at 0x1a421adcf8>




## the frequency of the age show that user from 18-33 are likely to grade the movies

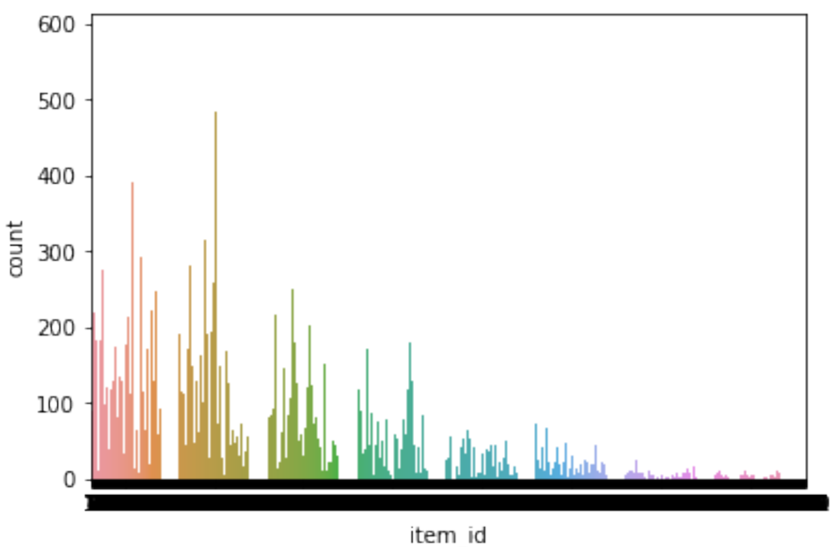
In [136]:
```
sns.countplot(x='user_id', data=ratings)
```

Out[136]: <matplotlib.axes._subplots.AxesSubplot at 0x1a40b570b8>



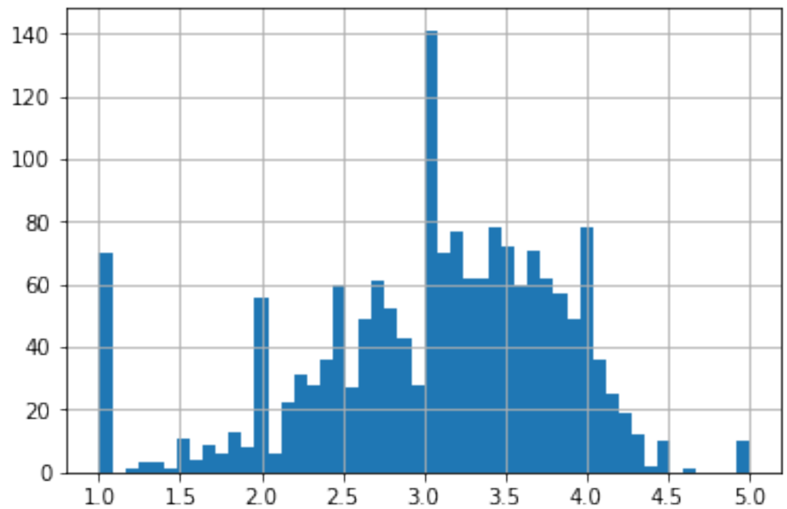

In [139]:
```
sns.countplot(x='item_id', data=ratings)
```

Out[139]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2fe18e48>



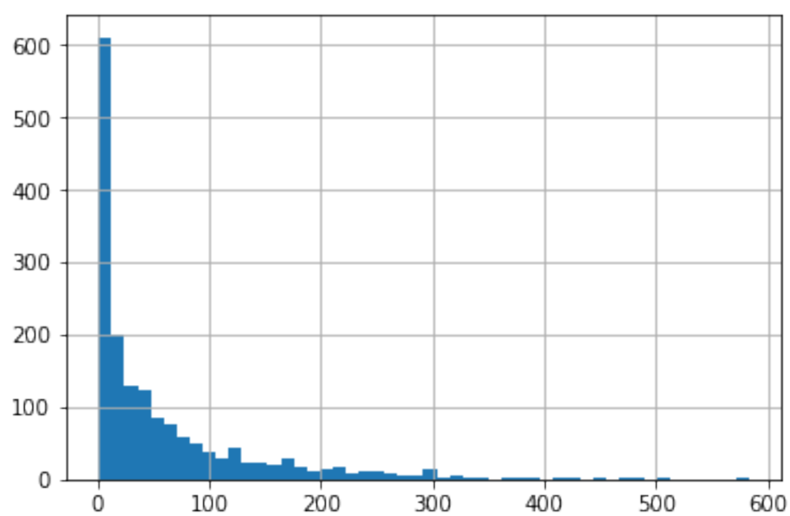
In [140]:
```
##soting the most popular items
ratings.groupby('item_id')['ratings'].mean().sort_values(ascending=False).hist(bins=50)
```

Out[140]: <matplotlib.axes._subplots.AxesSubplot at 0x1a33139c18>




In [141]:
```
##soting the most count items
ratings.groupby('item_id')['ratings'].count().sort_values(ascending=False).hist(bins=50)
```

Out[141]: <matplotlib.axes._subplots.AxesSubplot at 0x1a372ce0b8>




# Number of unrated is much higher

In [142]:
```
ratings.columns
```

Out[142]:
```
Index(['user_id', 'item_id', 'ratings', 'timestamp'], dtype='object'
)
```

In [143]:
```
tidy = ratings.drop('timestamp',axis=1).pivot_table(
index=['user_id'],
columns='item_id'
)
```

In [144]:
```
tidy.fillna(value=0., inplace=True)
tidy.head()
```

Out[144]:

| | ratings | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **item_id** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **...** | **1673** | **1674** | **1675** | **1676** | **1677** | **1678** |
| **user_id** | | | | | | | | | | | | | | | | | |
| **1** | 5.0 | 3.0 | 4.0 | 3.0 | 3.0 | 5.0 | 4.0 | 1.0 | 5.0 | 3.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 4.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 2.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **5** | 4.0 | 3.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

5 rows × 1682 columns

In [89]:
```
Normalizer = StandardScaler().fit
```

## Normailizeing the Data

In [ ]:
```
Normalizer = Normalizer().fit(R_matrix)
Normalizer.transform(R_matrix)
```

In [ ]:
```

```

## Setting the Model:

## Some helper functions:

fm_train_test_split

split the ratings matrix into train and test

Rating_Kfold

Perform cross-validation split on the given matrix

## Note: I use different plot for the test and the train, because there are in different scale

## I pick some random ratings from the ratings matrix and replace them by zeros/not rated and create a new Rating matrix/ test where all other are zeros excepte those pick in the Original rating matrix

In [92]:
```
def fm_train_test_split(Ratings, n_ratings=10):
    """
    Splite the Rating matrix into train and test
    n_rating: the number of non_empty ranting to assign to the test

    """
    test = np.zeros_like(Ratings) # the test matrix
    xs, ys = Ratings.nonzero()    # select the non zeros coordinate in
the Ratings
    index = np.arange(len(xs))
    train = Ratings.copy()
    np.random.shuffle(index)
    idx = index[:n_ratings]
    test[(xs[idx],ys[idx])] = Ratings[(xs[idx],ys[idx])]
    train[(xs[idx],ys[idx])] = 0.
    # Test and training are truly disjoint
    assert(np.all((train * test) == 0))
    return train, test

def Rating_Kfold(Ratings, n_fold=5):
    """
    Splite the Rating matrix into train and test
    n_rating: the number of non_empty ranting to assign to the test

    """
    xs, ys = Ratings.nonzero()    # select the non zeros coordinate in
the Ratings
    index = np.arange(len(xs))
    np.random.shuffle(index)
    batches = np.array_split(index, len(xs)//n_fold)
    for idx in batches:
        train = Ratings.copy()
        test = np.zeros_like(Ratings) # the test matrix
        test[(xs[idx],ys[idx])] = Ratings[(xs[idx],ys[idx])]
        train[(xs[idx],ys[idx])] = 0.
        assert(np.all((train * test) == 0))
        yield train, test
```

In [118]:
```
class SGD_MF():
    def __init__(self, R,test=np.array([None]), K=10, alpha=0.1, beta=
0.1, \
                 verbose=True, plot=False):

        self.R = R
        self.test = test
        self.num_users, self.num_items = R.shape
        self.K = K
        self.lr = alpha
        self.regul = beta
```

```
        self.verbose = verbose
        self.plot = plot

    def train(self, epoch=10):
        # Initialize user and item latent feature matrice
        self.U = np.random.uniform(low=0, high=1, size=(self.num_users
, self.K))
        self.V = np.random.uniform(low=0, high=1, size=(self.num_items
, self.K))

        # Initialize the biases
        self.user_bias = np.zeros(self.num_users)
        self.item_bias = np.zeros(self.num_items)
        self.global_bias = np.mean(self.R[np.where(self.R != 0)])

        # Create a list of training samples
        self.samples = [
            (i, j, self.R[i, j])
            for i in range(self.num_users)
            for j in range(self.num_items)
            if self.R[i, j] > 0
        ]

        # Perform stochastic gradient descent for number of iterations
        self.training_process = []
        self.test_process     = []
        for i in range(epoch):
            np.random.shuffle(self.samples)
            self.sgd()
            mse = self.rmse()
            self.training_process.append(mse)
            if self.test.any()!=None:
                test_mse = self.rmse(test=True)
                self.test_process.append(test_mse)
            if (i+1) % 10 == 0 and self.verbose:
                print(f"Iteration: {i} ;Train error = {mse}")
                if self.test.any()!=None:
                    print(f"Iteration: {i} ;test error = {test_mse}")
        if self.plot:
            self.plotting()
        return self.training_process, self.test_process

    def sgd(self):

        for x,i, r in self.samples:
            prediction = self.predict(x,i)
            e = (r - prediction)

            # Update biases
            self.user_bias[x] +=  self.lr * (e - self.regul* self.user
```

```
_bias[x])
            self.item_bias[i] +=  self.lr *(e - self.regul * self.item
_bias[i])

            #Update latent factors
            # I copy one of the latent factor sothat both get update s
imultaneously
            U_copy = self.U[x,:][:]
            self.U[x, :] +=   self.lr * (e * self.V[i, :] - self.regul
* self.U[x,:])
            self.V[i, :] +=   self.lr * (e * U_copy  - self.regul * se
lf.V[i,:])


    def predict(self, u, i):
        """
        Return the prediction for a given index
        """
        predict = self.global_bias + self.user_bias[u] + self.item_bia
s[i]
        predict = predict + self.U[u, :].dot(self.V[i, :].T)
        return predict

    def full_predict(self):
        """
        Predict the full ratings matrix
        """
        total_bias = self.global_bias + self.user_bias[:,np.newaxis]+
self.item_bias
        return total_bias + self.U.dot(self.V.T)

    def rmse(self, test=False):
        """
        return the mean square error
        if test: True return only for the test set
        """
        predicted = self.full_predict()
        if test:
            actual = self.test[self.test.nonzero()]
            pred = predicted[self.test.nonzero()]
            return np.sqrt(np.sum((pred - actual)**2))
        else:
            actual = self.R[self.R.nonzero()]
            pred = predicted[self.R.nonzero()]
            return np.sqrt(np.sum((pred - actual)**2))

    def plotting(self):
        fig, ax = plt.subplots(1,2,figsize=(15, 5))
        fig.suptitle('Matrix Factorization', fontsize=20)
        ax[0].grid()
```

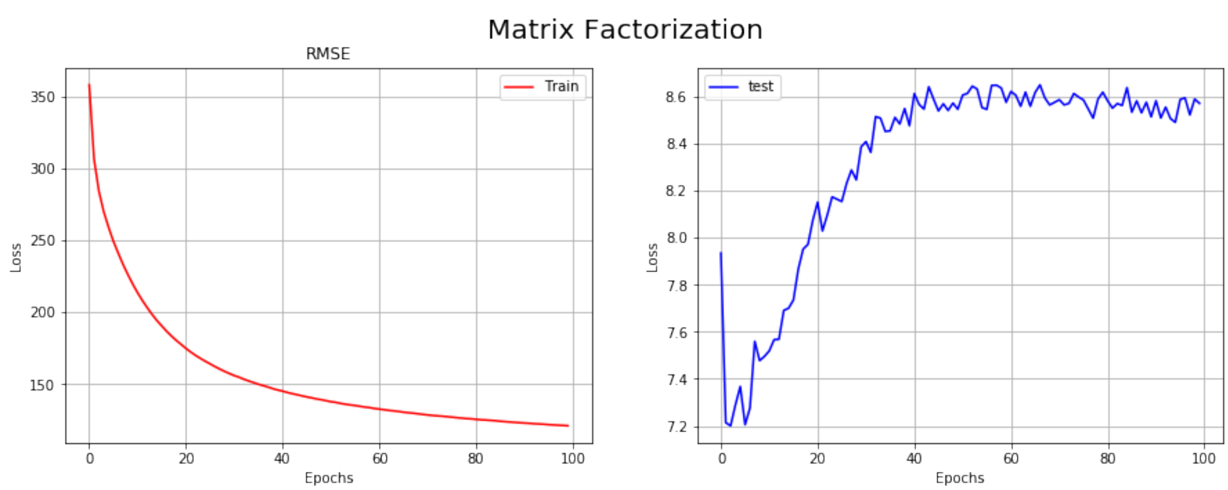
```
        ax[1].grid()
        ax[0].plot(self.training_process, label='Train',  color="r")
        if self.test.any()!= None:
            ax[1].plot(self.test_process,label='test',  color="b")
            ax[1].set_xlabel('Epochs')
            ax[1].set_ylabel('Loss')
            ax[1].legend()
        ax[0].set_title("RMSE")
        ax[0].set_xlabel('Epochs')
        ax[0].set_ylabel('Loss')
        ax[0].legend()

        plt.show()
```

In [120]:

```
train , test = fm_train_test_split(R_matrix, n_ratings=50)
mf = SGD_MF(train,test, K=40, alpha=0.01, beta=0.01, verbose=True, plo
t=True)
training_process = mf.train(epoch=100)
indeces = np.argmin(mf.test_process)
print('min_indece', indeces)
```

```
Iteration: 9 ;Train error = 219.68144495256502
Iteration: 9 ;test error = 7.495481322494891
Iteration: 19 ;Train error = 177.80798962182072
Iteration: 19 ;test error = 8.074121039441524
Iteration: 29 ;Train error = 157.52569090151485
Iteration: 29 ;test error = 8.386867803194786
Iteration: 39 ;Train error = 146.08319550918918
Iteration: 39 ;test error = 8.477260354428449
Iteration: 49 ;Train error = 138.65657925851647
Iteration: 49 ;test error = 8.547027526812624
Iteration: 59 ;Train error = 133.23381295096488
Iteration: 59 ;test error = 8.576703217059224
Iteration: 69 ;Train error = 129.15860148101356
Iteration: 69 ;test error = 8.576461090318812
Iteration: 79 ;Train error = 126.06216305538503
Iteration: 79 ;test error = 8.619841762463105
Iteration: 89 ;Train error = 123.3710064942707
Iteration: 89 ;test error = 8.51469430243873
Iteration: 99 ;Train error = 121.25840603469757
Iteration: 99 ;test error = 8.572174472000663
```




```
min_indece 2
```

We can the best performance is withing the 10 iterations

## evaluation and hyperparameter tuning

In [121]:
```
from itertools import product
train , test = fm_train_test_split(R_matrix, n_ratings=50)

regularization = [2, 5, 10, 25, 50, 100, 200]
learning_rates = [1e-3, 1e-2,1e-1]
latent_factor  = [x for x in range(10,100,10)]
epoch = 5
best_params = {}
best_params['learning_rate'] = None
best_params['latent_factor'] = None
best_params['n_iter'] = 0
best_params['train_mse'] = np.inf
best_params['test_mse'] = np.inf
best_params['model'] = None
it = 0
for regul, lr, K in product(regularization, learning_rates,latent_fact
or):
    MF_SGD = SGD_MF(train, test,K=K,alpha=lr,beta=regul,verbose=False)
    MF_SGD.train(epoch=epoch)
    min_idx = np.argmin(MF_SGD.test_process)
    if it>0 and MF_SGD.test_process[min_idx] < best_params['test_mse']
:
        best_params['n_iter'] = it
        best_params['learning_rate'] = lr
        best_params['latent_factor'] = K
        best_params['regularizer']   = regul
        best_params['train_mse'] = MF_SGD.training_process[min_idx]
        best_params['test_mse'] = MF_SGD.test_process[min_idx]
        best_params['model'] = MF_SGD
#          print(pd.Series(best_params))
    it +=1
```

```
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:65: RuntimeWarning: overflow encountered in multiply
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:64: RuntimeWarning: overflow encountered in multiply
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:64: RuntimeWarning: invalid value encountered in add
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:65: RuntimeWarning: invalid value encountered in sub
tract
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:58: RuntimeWarning: invalid value encountered in dou
ble_scalars
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:59: RuntimeWarning: invalid value encountered in dou
ble_scalars
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:73: RuntimeWarning: invalid value encountered in dou
ble_scalars
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:64: RuntimeWarning: invalid value encountered in mul
tiply
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:72: RuntimeWarning: invalid value encountered in dou
ble_scalars
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:64: RuntimeWarning: invalid value encountered in sub
tract
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:64: RuntimeWarning: overflow encountered in subtract
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:58: RuntimeWarning: overflow encountered in double_s
calars
/Users/souayboubagayoko/anaconda3/lib/python3.7/site-packages/ipyker
nel_launcher.py:59: RuntimeWarning: overflow encountered in double_s
calars
```

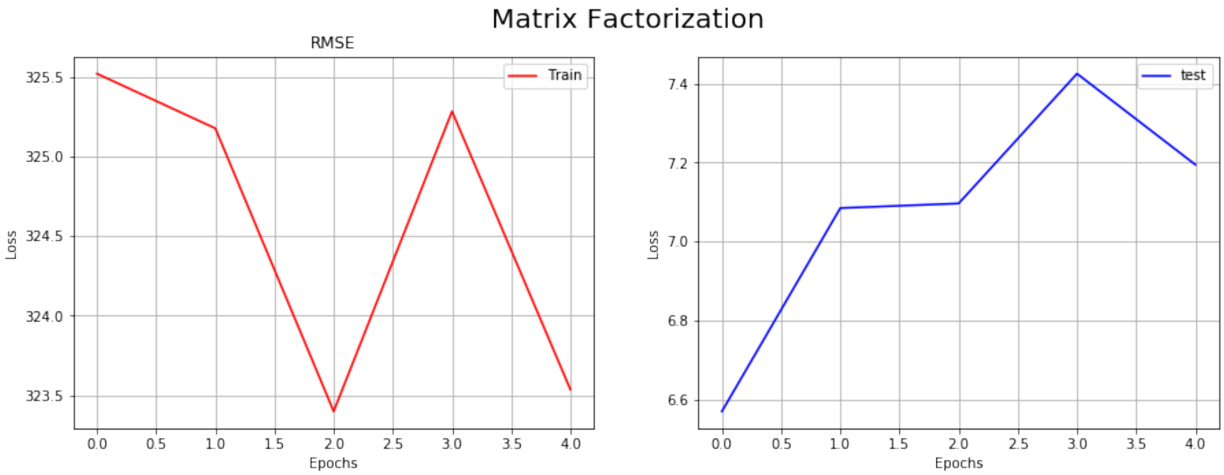
In [124]:

```
best_params
```

Out[124]:

```
{'learning_rate': 0.1,
 'latent_factor': 20,
 'n_iter': 19,
 'train_mse': 325.5187405536677,
 'test_mse': 6.569277461380019,
 'model': <__main__.SGD_MF at 0x1a2a92eb00>,
 'regularizer': 2}
```

In [125]:
```
best_params['model'].plotting()
```




## Exercise 3 :

In this Part I will use Sklearn

In [ ]:
```
from itertools import product
from sklearn.decomposition import NMF

def get_rmse(predict, actual):
    real = actual[actual.nonzero()]
    pred = predict[actual.nonzero()]
    return np.sqrt(np.sum((pred - real)**2))

#### defining the model ##############
model = NMF(init='random', solver='cd')

#######defining the Hyperparameters######
regularization =[1e-6,1e-5,1e-4,1e-3, 1e-2]
latent_factor  = [x for x in range(10,100,10)]
iteration  = [x for x in range(1,10,2)]
best_params = {}
best_params['latent_factor'] = None
best_params['n_iter'] = 0
best_params['test_mse'] = np.inf
best_params['model'] = None

Kfold = 3
###### cross validation #####
global_loss = []
it = 0
for train , test in Rating_Kfold(R_matrix, n_fold=Kfold):
    print(f"fold number {it+1}")

    loss_his = []
    for regul, max_iter, K in product(regularization, iteration,latent
_factor):
        model_param = {'n_components':K, 'max_iter':max_iter, 'alpha':
regul}
        model.set_params(**model_param)
        H = model.fit_transform(train)
        W = model.components_
        predict = H.dot(W)
        test_rmse = get_rmse(predict, test)
        loss_his.append(test_rmse)

    idx = np.argmin(loss_his)
    global_loss.append(loss_his)
    if  loss_his[idx] < best_params['test_mse']:
        best_params['n_iter'] = it
        best_params['latent_factor'] = K
        best_params['regularizer']   = regul
        best_params['test_mse'] = loss_his[idx]
        best_params['model'] = model
    it +=1
```

```
fold number 1
fold number 2
fold number 3
fold number 4
fold number 5
fold number 6
fold number 7
fold number 8
fold number 9
fold number 10
fold number 11
fold number 12
fold number 13
fold number 14
fold number 15
fold number 16
fold number 17
fold number 18
fold number 19
fold number 20
fold number 21
```

***show the best model comparing with the model in task 1 is not done due the time it take to runs I have not use for my model 3kfold best_params***

In [ ]:
```
best_params
```

In [ ]:
```
plt.plot(global_loss[2])
```

In [ ]:
```
## the Rining is taking too long
```

## Result:

#### In the above model:

#### solver : Coordinate Descent

#### H , W are the latent Factors

#### n_components: the latent number

#### alpha: The gularization parameter

#### frobenius norm: A generaliztion of L2 norm for the latent factors

In [ ]:

```

```