# SGD Algorithm to predict movie ratings

**There will be some functions that start with the word "grader" ex: grader_matrix(), grader_mean(), grader_dim() etc, you should not change those function definition.**

**Every Grader function has to return True.**

1. Download the data from here (https://drive.google.com/open?id=1-1z7iDB52cB6_JpO7Dqa-eOYSs-mivpq)
2. The data will be of this format, each data point is represented as a triplet of user_id, movie_id and rating

| user_id | movie_id | rating |
|---|---|---|
| 77 | 236 | 3 |
| 471 | 208 | 5 |
| 641 | 401 | 4 |
| 31 | 298 | 4 |
| 58 | 504 | 5 |
| 235 | 727 | 5 |

# Task 1

### Predict the rating for a given (user_id, movie_id) pair

Predicted rating $\hat{y}_{ij}$ for user i, movied j pair is calcuated as $\hat{y}_{ij} = \mu + b_i + c_j + u_i^T v_j$ , here we will be finding the best values of $b_i$ and $c_j$ using SGD algorithm with the optimization problem for N users and M movies is defined as

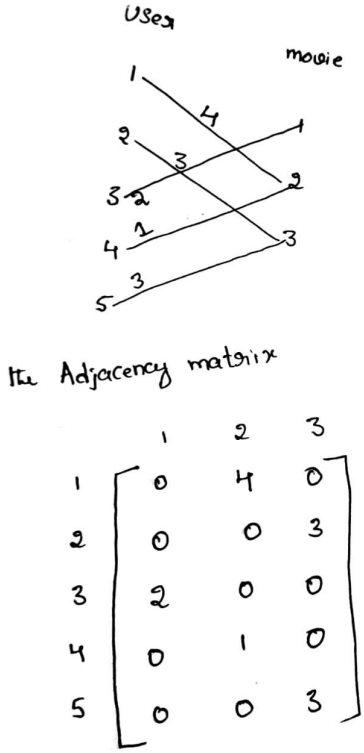
$$L = \min_{b,c,\{u_i\}_{i=1}^{N},\{v_j\}_{j=1}^{M}} \alpha\left(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2\right) + \sum_{i,j \in I^{\text{train}}} (y_{ij} - \mu - b_i - c_j - u_i^T v_j)^2$$

- (\mu) : scalar mean rating
- (b_i) : scalar bias term for user (i)
- (c_j) : scalar bias term for movie (j)
- (u_i) : K-dimensional vector for user (i)
- (v_j) : K-dimensional vector for movie (j)

*. We will be giving you some functions, please write code in that functions only.

*. After every function, we will be giving you expected output, please make sure that you get that output.

1. Construct adjacency matrix with the given data, assuming its graph and the weight of each edge is the rating given by user to the movie




The Adjacency matrix

| | 1 | 2 | 3 |
|---|---|---|---|
| 1 | 0 | 4 | 0 |
| 2 | 0 | 0 | 3 |
| 3 | 2 | 0 | 0 |
| 4 | 0 | 1 | 0 |
| 5 | 0 | 0 | 3 |

you can construct this matrix like $A[i][j] = r_{ij}$ here $i$ is user_id, $j$ is movie_id and $r_{ij}$ is rating given by user $i$ to the movie $j$

Hint : you can create adjacency matrix using csr_matrix (https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html)

2. We will Apply SVD decomposition on the Adjaceny matrix link1 (https://stackoverflow.com/a/31528944/4084039), link2 (https://machinelearningmastery.com/singular-value-decomposition-for-machine-learning/) and get three matrices $U$, $\sum$, $V$ such that $U \times \sum \times V^T = A$, if $A$ is of dimensions $N \times M$ then
U is of $N \times k$,
$\sum$ is of $k \times k$ and
$V$ is $M \times k$ dimensions.

   *. So the matrix $U$ can be represented as matrix representation of users, where each row $u_i$ represents a k-dimensional vector for a user

   *. So the matrix $V$ can be represented as matrix representation of movies, where each row $v_j$ represents a k-dimensional vector for a movie.
3. Compute $\mu$ , $\mu$ represents the mean of all the rating given in the dataset.(write your code in def m_u())
4. For each unique user initilize a bias value $B_i$ to zero, so if we have $N$ users $B$ will be a $N$ dimensional vector, the $i^{th}$ value of the $B$ will corresponds to the bias term for $i^{th}$ user (write your code in def initialize())
5. For each unique movie initilize a bias value $C_j$ zero, so if we have $M$ movies $C$ will be a $M$ dimensional vector, the $j^{th}$ value of the $C$ will corresponds to the bias term for $j^{th}$ movie (write your code in def initialize())
6. Compute dL/db_i (Write you code in def derivative_db())

7. Compute dL/dc_j(write your code in def derivative_dc()
8. Print the mean squared error with predicted ratings.

```
for each epoch:
    for each pair of (user, movie):
        b_i =  b_i - learning_rate  dL/db_i
        c_j =  c_j - learning_rate  dL/dc_j
predict the ratings with formula
```

$$\hat{y}_{ij} = \mu + b_i + c_j + \text{dot\_product}(u_i, v_j)$$

9. you can choose any learning rate and regularization term in the range $10^{-3}$ to $10^{2}$
10. **bonus**: instead of using SVD decomposition you can learn the vectors $u_i$, $v_j$ with the help of SGD algo similar to $b_i$ and $c_j$

In [1]:

```
import networkx as nx
from networkx.algorithms import bipartite
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
# you need to have tensorflow
from stellargraph.data import UniformRandomMetaPathWalk
from stellargraph import StellarGraph
```

# Task 2

As we know U is the learned matrix of user vectors, with its i-th row as the vector ui for user i. Each row of U can be seen as a "feature vector" for a particular user.

The question we'd like to investigate is this: do our computed per-user features that are optimized for predicting movie ratings contain anything to do with gender?

The provided data file user_info.csv (https://drive.google.com/open?id=1PHFdJh_4gIPiLH5Q4UErH8GK71hTrzIY) contains an is_male column indicating which users in the dataset are male. Can you predict this signal given the features U?

> **Note 1** : there is no train test split in the data, the goal of this assignment is to give an intution about how to do matrix factorization with the help of SGD and application of truncated SVD. for better understanding of the collabarative fillerting please check netflix case study.
>
> **Note 2** : Check if scaling of $U$, $V$ matrices improve the metric

## Reading the csv file

In [69]:

```
import pandas as pd
data=pd.read_csv('ratings_train.csv')
data.head()
```

Out[69]:

| | user_id | item_id | rating |
|---|---|---|---|
| **0** | 772 | 36 | 3 |
| **1** | 471 | 228 | 5 |
| **2** | 641 | 401 | 4 |
| **3** | 312 | 98 | 4 |
| **4** | 58 | 504 | 5 |

In [70]:

```
data.shape
```

Out[70]:

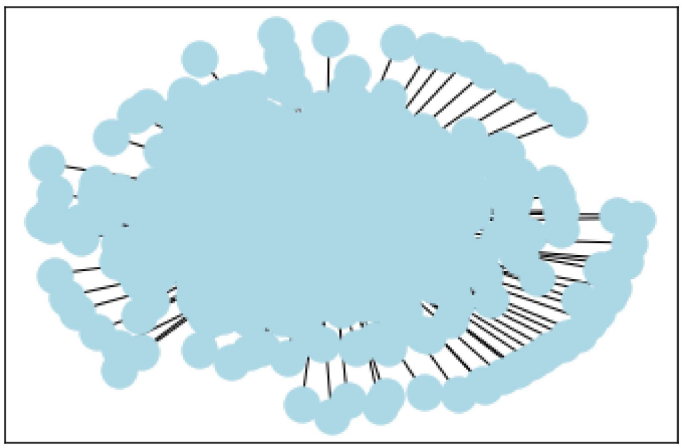
```
(89992, 3)
```

In [71]:

```
edges = [tuple(x) for x in data.values.tolist()]
```

In [5]:

```
B = nx.Graph()
B.add_nodes_from(data['user_id'].unique(), bipartite=0, label='user')
B.add_nodes_from(data['item_id'].unique(), bipartite=1, label='movie')
B.add_weighted_edges_from(edges)
nx.draw_networkx(B,node_color='lightblue',with_labels=False)
```



## Create your adjacency matrix

In [72]:

```
adjacency_matrix = csr_matrix((list(data['rating'].values), (list(data['user_id'].values),
```

In [73]:

```
adjacency_matrix.shape
```

Out[73]:

```
(943, 1681)
```

Grader function - 1

In [74]:

```
def grader_matrix(matrix):
  assert(matrix.shape==(943,1681))
  return True
grader_matrix(adjacency_matrix)
```

Out[74]:

```
True
```

**The unique items in the given csv file are 1662 only . But the id's vary from 0-1681 but they are not continuous and hence you'll get matrix of size 943x1681.**

SVD decompostion

Sample code for SVD decompostion

In [75]:

```
from sklearn.utils.extmath import randomized_svd
import numpy as np
matrix = np.random.random((20, 10))
U, Sigma, VT = randomized_svd(matrix, n_components=5,n_iter=5, random_state=None)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(20, 5)
(5,)
(10, 5)
```

Write your code for SVD decompostion

In [76]:

```
# Please use adjacency_matrix as matrix for SVD decompostion
# You can choose n_components as your choice
U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=5,n_iter=5, random_state=None)
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

```
(943, 5)
(5,)
(1681, 5)
```

Compute mean of ratings

In [77]:

```
import pandas as pd
def m_u(ratings):
    '''In this function, we will compute mean for all the ratings'''
    # you can use mean() function to do this
    # check this (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFra
    mean_val = ratings.mean()
    return mean_val
```

In [78]:

```
mu=m_u(data['rating'])
print(mu)
```

```
3.529480398257623
```

Grader function -2

In [79]:

```
def grader_mean(mu):
  assert(np.round(mu,3)==3.529)
  return True
mu=m_u(data['rating'])
grader_mean(mu)
```

Out[79]:

```
True
```

Initialize $B_i$ and $C_j$

Hint : Number of rows of adjacent matrix corresponds to user dimensions($B_i$), number of columns of adjacent matrix corresponds to movie dimensions ($C_j$)

In [80]:

```
def initialize(dim):
    '''In this function, we will initialize bias value 'B' and 'C'.'''
    # initalize the value to zeros
    # return output as a list of zeros
    init_lst = []
    for i in range(dim):
        init_lst.append(0)
    return init_lst
```

In [81]:

```
dim= adjacency_matrix.shape[0] # give the number of dimensions for b_i (Here b_i correspond
b_i=initialize(dim)
mve_dim = adjacency_matrix.shape[1]
c_j = initialize(mve_dim)
```

Grader function -3

In [82]:

```
def grader_dim(b_i,c_j):
  assert(len(b_i)==943 and np.sum(b_i)==0)
  assert(len(c_j)==1681 and np.sum(c_j)==0)
  return True
grader_dim(b_i,c_j)
```

Out[82]:

```
True
```

Compute dL/db_i

In [83]:

```
def derivative_db(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/db_i'''
    dL_dbi = 2*(b_i[user_id]*(alpha+1)+mu+c_j[item_id]+np.matmul(U[user_id],V[:,item_id])-r
    return dL_dbi
```

Grader function -4

In [84]:

```
def grader_db(value):
    assert(np.round(value,3)==-0.931)
    return True
U1, Sigma, V1 = randomized_svd(adjacency_matrix, n_components=2,n_iter=5, random_state=24)
# Please don't change random state
# Here we are considering n_componets = 2 for our convinence
alpha=0.01
value=derivative_db(312,98,4,U1,V1,mu,alpha)
grader_db(value)
```

Out[84]:

```
True
```

Compute dL/dc_j

In [85]:

```
def derivative_dc(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/dc_j'''
    dL_dcj = 2*(c_j[item_id]*(alpha+1)+mu+b_i[user_id]+np.matmul(U[user_id],V[:,item_id])-r
    return dL_dcj
```

Grader function - 5

In [86]:

```
def grader_dc(value):
    assert(np.round(value,3)==-2.929)
    return True
U1, Sigma, V1 = randomized_svd(adjacency_matrix, n_components=2,n_iter=5, random_state=24)
# Please don't change random state
# Here we are considering n_componets = 2 for our convinence
alpha=0.01
value=derivative_dc(58,504,5,U1,V1,mu,alpha)
grader_dc(value)
```

Out[86]:

```
True
```

Compute MSE (mean squared error) for predicted ratings

for each epoch, print the MSE value

```
    for each epoch:

        for each pair of (user, movie):

            b_i =  b_i - learning_rate * dL/db_i

            c_j =  c_j - learning_rate * dL/dc_j
```

predict the ratings with formula

$$\hat{y}_{ij} = \mu + b_i + c_j + \text{dot\_product}(u_i, v_j)$$

In [87]:

```
from tqdm import tqdm
from sklearn.metrics import mean_squared_error

loss_lst = []
alpha = 0.001
learning_rate = 0.001
epochs = 30

def pred():
    y_predict = []
    for u,i,r in list(zip(data['user_id'].values, data['item_id'].values, data['rating'].va
        y_val = mu+b_i[u]+c_j[i]+np.matmul(U[u].transpose(),VT[:,i])
        y_predict.append(y_val)
    return y_predict

MSE_lst = []
for j in tqdm(range(epochs)):
    for u,i,r in list(zip(data['user_id'].values, data['item_id'].values, data['rating'].va
        dL_dbi = derivative_db(u,i,r,U,VT,mu,alpha)
        dL_dcj = derivative_dc(u,i,r,U,VT,mu,alpha)
        b_i[u] =  b_i[u]-learning_rate*dL_dbi
        c_j[i] =  c_j[i]-learning_rate*dL_dcj
    y_actual = rating_arr.tolist()
    y_predicted = pred()
    min_loss_val = mean_squared_error(y_actual, y_predicted)
    MSE_lst.append(min_loss_val)
print(MSE_lst)
```

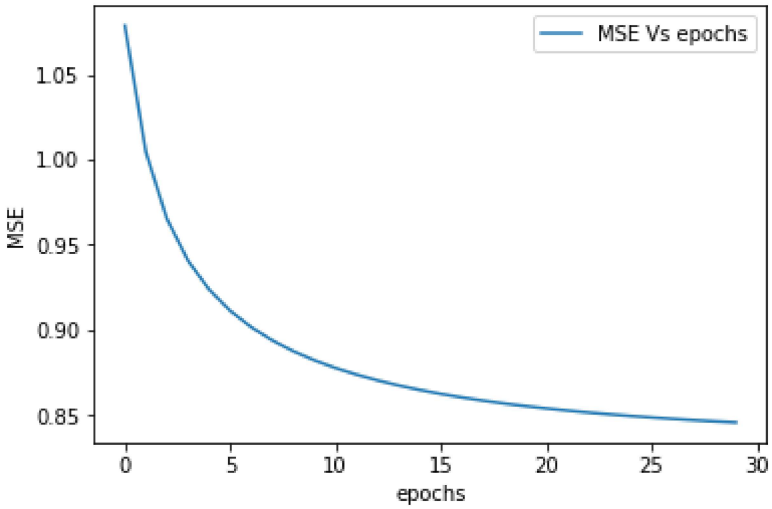
```
100%|██████████| 30/30 [00:56<00:00,  1.90s/it]

[1.0785842209106935, 1.0045082871368167, 0.9652414475795135, 0.9407670314248
497, 0.9238890095742731, 0.9114315027803409, 0.901788851159944, 0.8940643262
512454, 0.8877167914368694, 0.8823987115687224, 0.8778752311090237, 0.873980
8382218879, 0.8705947648773522, 0.8676263078645324, 0.8650056815371351, 0.86
26780969810011, 0.8605997992185654, 0.8587353353705461, 0.8570556214767576,
0.8555365423789015, 0.8541579165878307, 0.8529027168795664, 0.85175647384383
82, 0.8507068128129267, 0.8497430897052067, 0.8488561013654208, 0.8480378528
002127, 0.8472813684176517, 0.8465805376903377, 0.8459299880262362]
```

Plot epoch number vs MSE

- epoch number on X-axis
- MSE on Y-axis

In [88]:

```
plt.plot(list(range(epochs)),MSE_lst,label="MSE Vs epochs")
plt.xlabel("epochs")
plt.ylabel("MSE")
plt.legend()
plt.show()
```




# Task 2

- For this task you have to consider the user_matrix U and the user_info.csv file.
- You have to consider is_male columns as output features and rest as input features. Now you have to fit a model by posing this problem as binary classification task.
- You can apply any model like Logistic regression or Decision tree and check the performance of the model.
- Do plot confusion matrix after fitting your model and write your observations how your model is performing in this task.
- Optional work- You can try scaling your U matrix.Scaling means changing the values of n_componenets while performing svd and then check your results.

In [49]:

```
User_data=pd.read_csv('user_info.csv')
User_data.head()
```

Out[49]:

| | user_id | age | is_male | orig_user_id |
|---|---|---|---|---|
| **0** | 0 | 24 | 1 | 1 |
| **1** | 1 | 53 | 0 | 2 |
| **2** | 2 | 23 | 1 | 3 |
| **3** | 3 | 24 | 1 | 4 |
| **4** | 4 | 33 | 0 | 5 |

In [50]:

```
y = User_data['is_male']
X = User_data.drop(columns=['user_id','is_male'])
```

In [51]:

```
X.shape, y.shape
```

Out[51]:

```
((943, 2), (943,))
```

In [52]:

```
U.shape
```

Out[52]:

```
(943, 5)
```

In [53]:

```
from scipy.sparse import hstack
import scipy as scipy

X = hstack((U,scipy.sparse.csr_matrix(X))).tocsr()
```

In [54]:

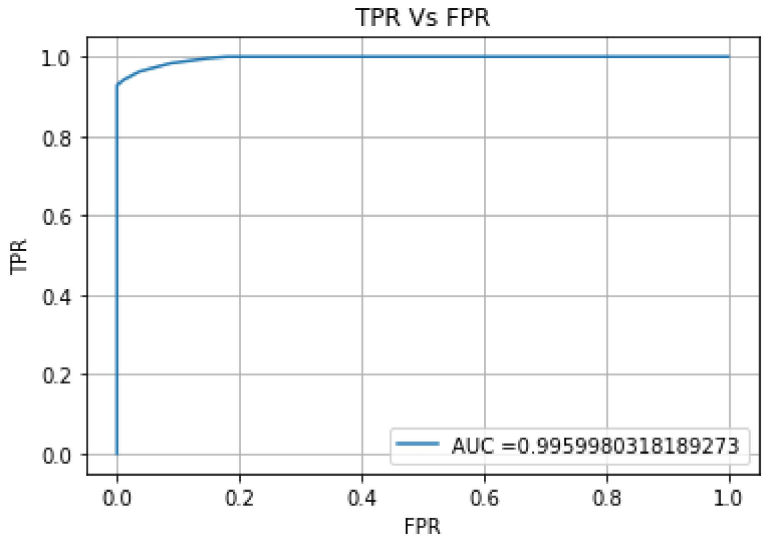
```
X.shape
```

Out[54]:

```
(943, 7)
```

In [55]:

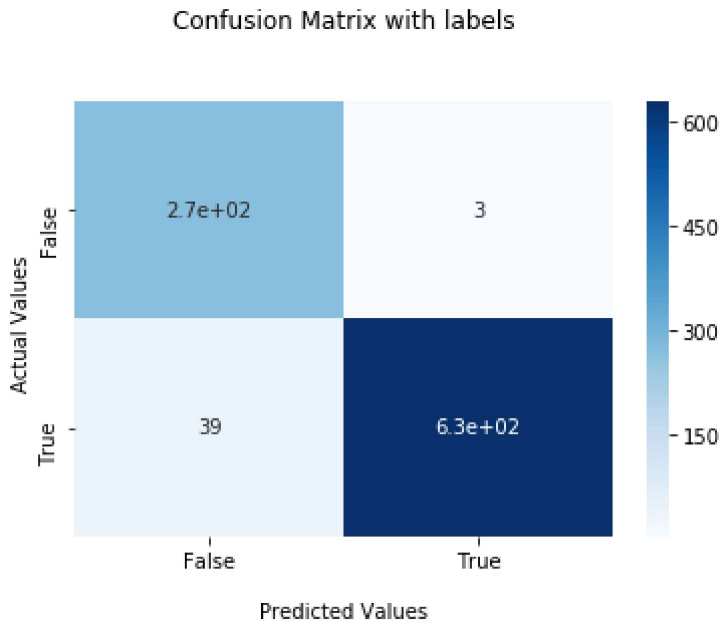
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint as sp_randint
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
import pandas as pd
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
from sklearn.metrics import roc_curve, auc

def batch_predict(clf, data):
    # roc_auc_score(y_true, y_score) the 2nd parameter should be probability estimates of t
    # not the predicted outputs

    y_data_pred = []
    tr_loop = data.shape[0] - data.shape[0]%1000
    # consider you X_tr shape is 49041, then your tr_loop will be 49041 - 49041%1000 = 4900
    # in this for loop we will iterate unti the last 1000 multiplier
    for i in range(0, tr_loop, 1000):
        y_data_pred.extend(clf.predict_proba(data[i:i+1000])[:,1])
    # we will be predicting for the last data points
    if data.shape[0]%1000 !=0:
        y_data_pred.extend(clf.predict_proba(data[tr_loop:])[:,1])

    return y_data_pred

clf_tfidf = DecisionTreeClassifier(max_depth = 30, min_samples_split = 5, criterion='gini',
clf_tfidf.fit(X, y)
y_trn_predict = batch_predict(clf_tfidf, X)
train_fpr, train_tpr, tr_thresholds = roc_curve(y, y_trn_predict)
set1_tr_AUC = auc(train_fpr, train_tpr)
plt.plot(train_fpr, train_tpr, label="AUC ="+str(set1_tr_AUC))
plt.legend()
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("TPR Vs FPR")
plt.grid()
plt.show()
```


TPR Vs FPR
TPR
FPR
AUC =0.9959980318189273
0.0
0.2
0.4
0.6
0.8
1.0

In [56]:

```
# we are writing our own function for predict, with defined thresould
# we will pick a threshold that will give the least fpr
def find_best_threshold(threshould, fpr, tpr):
    t = threshould[np.argmax(tpr*(1-fpr))]
    # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high
    print("the maximum value of tpr*(1-fpr)", max(tpr*(1-fpr)), "for threshold", np.round(t
    return t

def predict_with_best_t(proba, threshould):
    predictions = []
    for i in proba:
        if i>=threshould:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions

from sklearn.metrics import confusion_matrix
best_t = find_best_threshold(tr_thresholds, train_fpr, train_tpr)

# The below code is refered from the following link: https://www.stackvidhya.com/plot-confu

import seaborn as sns

ax = sns.heatmap(confusion_matrix(y, predict_with_best_t(y_trn_predict, best_t)), annot=Tru

ax.set_title('Confusion Matrix with labels\n\n');
ax.set_xlabel('\nPredicted Values')
ax.set_ylabel('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False','True'])
ax.yaxis.set_ticklabels(['False','True'])

## Display the visualization of the Confusion Matrix.
plt.show()
```

```
the maximum value of tpr*(1-fpr) 0.9314416926357225 for threshold 0.75
```



```
In this case, seems like model is overfitted
```