Recommendation_System_TruncatedSVD

October 22, 2021

1 SGD Algorithm to predict movie ratings

1.1 Task 1

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

Reading the csv file

[1]:
```
import pandas as pd
data=pd.read_csv('ratings_train.csv')
data.head()
```

[1]:
```
   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
```

[2]:
```
data.shape
```

[2]:
```
(89992, 3)
```

Creating adjacency matrix

[3]:
```
from scipy.sparse import csr_matrix
adjacency_matrix␣
↪=csr_matrix((data['rating'],(data['user_id'],data['item_id'])),dtype=int) #␣
↪write your code of adjacency matrix here
```

[4]:
```
adjacency_matrix.shape
```

[4]:
```
(943, 1681)
```

Grader function - 1

[5]:
```
def grader_matrix(matrix):
  assert(matrix.shape==(943,1681))
  return True
grader_matrix(adjacency_matrix)
```

[5]: True

SVD decompostion

[6]:
```
# using adjacency_matrix for SVD decompostion

from sklearn.utils.extmath import randomized_svd
import numpy as np

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

[7]:
```
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.DataFrame.mean.html) link for more details.
    return ratings.mean()
```

[8]:
```
mu=m_u(data['rating'])
print(mu)
```

```
3.529480398257623
```

Grader function -2

[9]:
```
def grader_mean(mu):
  assert(np.round(mu,3)==3.529)
  return True
mu=m_u(data['rating'])
grader_mean(mu)
```

[9]: True

Initialize B_i and C_j

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

[10]:
```
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
    return np.zeros(dim)
```

[11]:
```
dim= adjacency_matrix.shape[0]# giving the number of dimensions for b_i (Here␣
↪b_i corresponds to users)
b_i=initialize(dim)
```

[12]:
```
dim= adjacency_matrix.shape[1]# giving the number of dimensions for c_j (Here␣
↪c_j corresponds to movies)
c_j=initialize(dim)
```

Grader function -3

[13]:
```
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)
```

[13]:
```
True
```

Compute dL/db_i

[14]:
```
def derivative_db(user_id,item_id,rating,U,V,mu,alpha):
    '''In this function, we will compute dL/db_i'''
    res=alpha*(2*b_i[user_id])+2*(rating-mu-b_i[user_id]-c_j[item_id]-np.
↪dot(U[user_id],V.T[item_id]))*(-1)
    return res
```

Grader function -4

[15]:
```
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)

# 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)
```

[15]:
```
True
```

Compute dL/dc_j

```
[16]: def derivative_dc(user_id,item_id,rating,U,V,mu):
          '''In this function, we will compute dL/dc_j'''
          res=alpha*(2*c_j[item_id])+2*(rating-mu-b_i[user_id]-c_j[item_id]-np.
       ↪dot(U[user_id],V.T[item_id]))*(-1)
          return res
```

Grader function - 5

```
[17]: 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)

      # Here we are considering n_componets = 2 for our convinence
      r=0.01
      value=derivative_dc(58,504,5,U1,V1,mu)
      grader_dc(value)
```

```
[17]: True
```

Compute MSE (mean squared error) for predicted ratings

```
[18]: y=np.array(data['rating'])
      user_id=np.array(data['user_id'])
      item_id=np.array(data['item_id'])
```

```
[19]: epoch=50
      learning_rate=0.001
      MSE=[]
      curr_mse=0
      length=len(data)
      for i in range(1,epoch): # each epoch
          y_pred=[]
          prev_mse=curr_mse
          for index in range(length): # for each user and movie
              userid=user_id[index]
              itemid=item_id[index]
              ␣
       ↪b_i[userid]=b_i[userid]-learning_rate*derivative_db(userid,itemid,y[index],U,VT,mu,alpha)␣
       ↪#updating user bais
              ␣
       ↪c_j[itemid]=c_j[itemid]-learning_rate*derivative_dc(userid,itemid,y[index],U,VT,mu)␣
       ↪#updating movie bais
          for ind in range(length):
              userid=user_id[ind]
              itemid=item_id[ind]
```

```
        ypred=mu+b_i[userid]+c_j[itemid]+np.dot(U[userid],VT.T[itemid])␣
↪#predicting rating
        y_pred.append(ypred)
    y_pred=np.array(y_pred)
    curr_mse=np.mean(np.square(y-y_pred)) #calculating mean squared error
    print("at epoch ",i," MSE=",curr_mse)
    if np.round(curr_mse,3)==np.round(prev_mse,3):
        epoch_value=i
        break
    MSE.append(curr_mse)
```

```
at epoch  1  MSE= 1.0789504830689258
at epoch  2  MSE= 1.0050283469912962
at epoch  3  MSE= 0.9657973842122543
at epoch  4  MSE= 0.9413218441323338
at epoch  5  MSE= 0.924432128471976
at epoch  6  MSE= 0.9119611421001191
at epoch  7  MSE= 0.9023060448416521
at epoch  8  MSE= 0.8945707810866832
at epoch  9  MSE= 0.8882141270920174
at epoch  10  MSE= 0.882888231555782
at epoch  11  MSE= 0.8783578964442278
at epoch  12  MSE= 0.8744573121608594
at epoch  13  MSE= 0.8710654755863043
at epoch  14  MSE= 0.8680915084953886
at epoch  15  MSE= 0.8654655008991121
at epoch  16  MSE= 0.8631325796497138
at epoch  17  MSE= 0.8610489360677697
at epoch  18  MSE= 0.8591790861510693
at epoch  19  MSE= 0.8574939310896514
at epoch  20  MSE= 0.8559693522877421
at epoch  21  MSE= 0.8545851725570873
at epoch  22  MSE= 0.8533243739799534
at epoch  23  MSE= 0.8521724994546295
at epoch  24  MSE= 0.8511171881794503
at epoch  25  MSE= 0.8501478104737527
at epoch  26  MSE= 0.8492551774119711
at epoch  27  MSE= 0.8484313075879562
at epoch  28  MSE= 0.8476692380560537
```

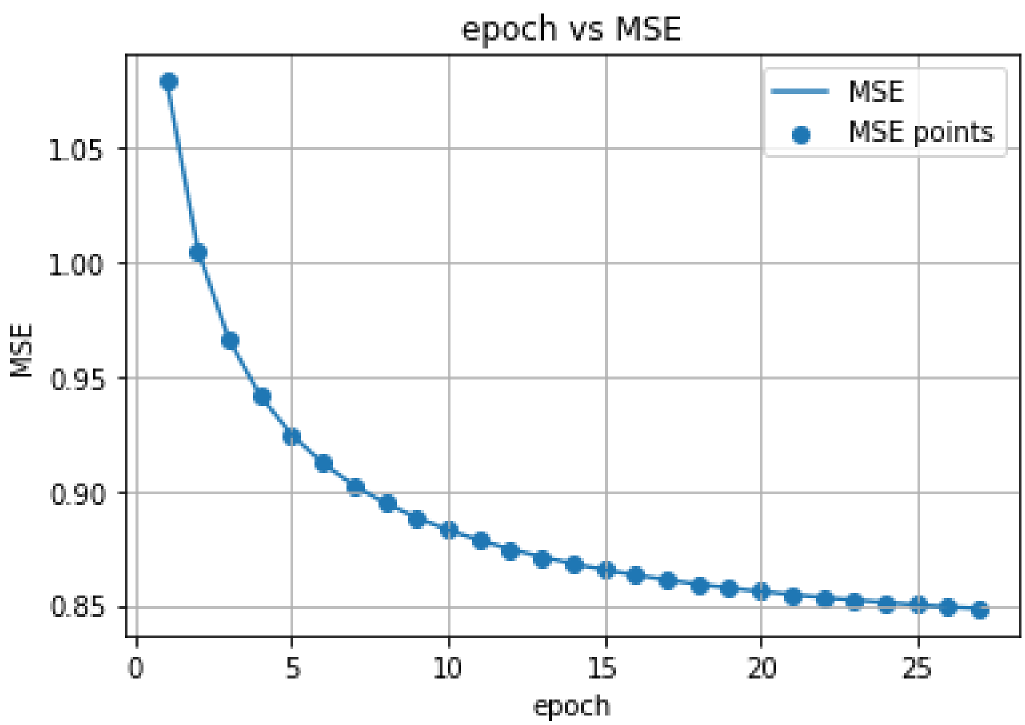
Plotting epoch number vs MSE

[28]:
```
#plotting MSE for each epoch
import matplotlib.pyplot as plt

epoch=range(1,epoch_value)

plt.plot(epoch,MSE,label='MSE')
```

```
plt.scatter(epoch,MSE,label='MSE points')
plt.legend()
plt.xlabel("epoch")
plt.ylabel("MSE")
plt.title("epoch vs MSE")
plt.grid()
plt.show()
```



Task 2

[21]:
```
user=pd.read_csv('user_info.csv.txt') #reading csv
```

[22]:
```
user.head()
```

[22]:

	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

[23]:
```
X=U # user features
y=user['is_male'] #target variable
```

[24]:
```
length=len(user)
```

[43]:
```
#applying logisticRegression
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression().fit(X, y)
```

[44]:
```
pred=reg.predict(X)
```

[51]:
```
from sklearn.metrics import confusion_matrix
C=confusion_matrix(y,pred)
print(C)
```

```
[[  0 273]
 [  0 670]]
```

After scaling user features

[54]:
```
from sklearn import preprocessing
```

[55]:
```
ss= preprocessing.StandardScaler()
X_scaling=ss.fit_transform(X)
```

[56]:
```
#applying logisticRegression

reg = LogisticRegression().fit(X_scaling, y)
pred=reg.predict(X)
```

[57]:
```
C=confusion_matrix(y,pred)
print(C)
```

```
[[  0 273]
 [  0 670]]
```

[66]:
```
user['is_male'].value_counts()
```

[66]:
```
1    670
0    273
Name: is_male, dtype: int64
```

Here if you observe the confusion matrix values of the model didn't change after scaling of user features From the confusion matrix we can says all the users with class label 1 are correctly predicted as 1 and all users with class label 0 are incorrectly predicated as 1 i.e all the users either with class label 1 or 0 are predicated as 1 We can conclude that the user features computed are not helpful in predicting the gender of the user

[ ]:
```

```