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 
2. The data will be of this format, each data point is represented as a triplet of user_
```

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

Saved successfully!

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

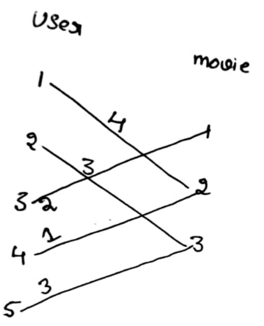
$$L = \min_{b, c, \{u_i\}_{i=1}^N, \{v_j\}_{j=1}^M} \alpha \Big(\sum_j \sum_k v_{jk}^2 + \sum_i \sum_k u_{ik}^2 + \sum_i b_i^2 + \sum_j c_i^2 \Big) + \sum_{i,j \in \mathcal{I}^{\text{train}}} (y_{ij} - \mu$$

- μ : 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

$$
\begin{array}{c|c}
 & \begin{matrix} 1 & 2 & 3 \end{matrix} \\
\begin{matrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 4 & 0 \\ 0 & 0 & 3 \\ 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}
\end{array}
$$

$j] = r_{ij}$ 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]

2. We will Apply SVD decomposition on the Adjaceny matrix [link1], [link2] 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 μ , μ 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

Saved successfully! ×

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 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

```
from google.colab import files
files= files.upload()
```

Choose Files ratings_train.csv

- **ratings_train.csv**(application/vnd.ms-excel) - 880367 bytes, last modified: 7/28/2019 - 100% done

```
Saving ratings_train.csv to ratings_train.csv
```

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

	user_id	item_id	rating
0	772	36	3
[illegible]	[illegible]	[illegible]	[illegible]
2	[illegible]	[illegible]	[illegible]
3	312	98	4
4	58	504	5

Saved successfully!

```
data.shape
```

```
(89992, 3)
```

```
data.describe()
```

	user_id	item_id	rating
count	89992.000000	89992.000000	89992.000000
mean	461.579151	423.584663	3.529480
75%	681.000000	629.000000	4.000000

```
User_ID= data['user_id']
Item_ID= data['item_id']
Rating=data['rating']
```

Create your adjacency matrix

```
from scipy.sparse import csr_matrix
import numpy as np
import scipy
# write your code of adjacency matrix here
row= data['user_id'].values
col= data['item_id'].values
rat= data['rating'].values

adjacency_matrix = csr_matrix((rat, (row, col)), shape=(943,1681)).toarray()
```

```
adjacency_matrix.shape
```

```
(943, 1681)
```

Saved successfully!

Grader function - 1

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

```
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

```
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

```
# Please use adjacency_matrix as matrix for SVD decompostion
# You can choose n_components as your choice

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

U, Sigma, VT = randomized_svd(adjacency_matrix, n_components=5, n_iter=5, random_state=Non
print(U.shape)
print(Sigma.shape)
print(VT.T.shape)
```

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

Compute mean of ratings

Saved successfully!

```
def m_u(ratings):
  mean_of_ratings = data["rating"].mean()
  return mean_of_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.DataFr
```

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

```
3.529480398257623
```

Grader function -2

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

```
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)

```
import numpy as np
#'''In this function, we will initialize bias value 'B' and 'C'.'''
def initialize(dim):
  B = np.zeros(dim)
  C = np.zeros(dim)
  return B.tolist() , C.tolist()

  # initalize the value to zeros
  # return output as a list of zeros
```

```
dim=943  #give the number of dimensions for b_i (Here b_i corresponds to users)
b_i,c_i = initialize(dim)
```

```
len(b_i)
```

```
943
```

```
dim=1681 #give the number of dimensions for c_j (Here c_j corresponds to movies)
```

Saved successfully!

```
len(c_j)
```

```
1681
```

Grader function -3

```
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)
```

```
True
```

Compute dL/db_i

```
def derivative_db(user_id,item_id,rating,U,V,mu,alpha): #'''In this function, we will comp
  db=2*alpha*(b_i[user_id])-2*(rating-mu-b_i[user_id]-c_j[item_id]-np.dot(U[user_id],V[:,i
  return db
```

Grader function -4

```
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)
```

```
True
```

Compute dL/dc_j

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

Grader function - 5


```
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
r=0.01
value=derivative_dc(58,504,5,U1,V1,mu,alpha)
grader_dc(value)
```

```
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

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

```
U_i= np.array(data['user_id'])
V_j= np.array(data['item_id'])
```

```
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

for a in range(0,11):
  summation= 0
  y_pred=[]
  y=[]
  learning_rate = 0.1
  for i,j,k in zip(User_ID, Item_ID, Rating):
    b_i[i]= b_i[i] - learning_rate * derivative_db(i,j,k,U1,V1,mu,alpha)
    c_j[j]= c_j[j] - learning_rate * derivative_dc(i,j,k,U1,V1,mu,alpha)

    y_i_j = (mu + b_i[i] + c_j[j] + np.dot(U_i[User_ID], V_j[Item_ID]))
```



```
ed)
  print("Epoch Number = ", a+1, "The mean square error is =  ",mse)
```

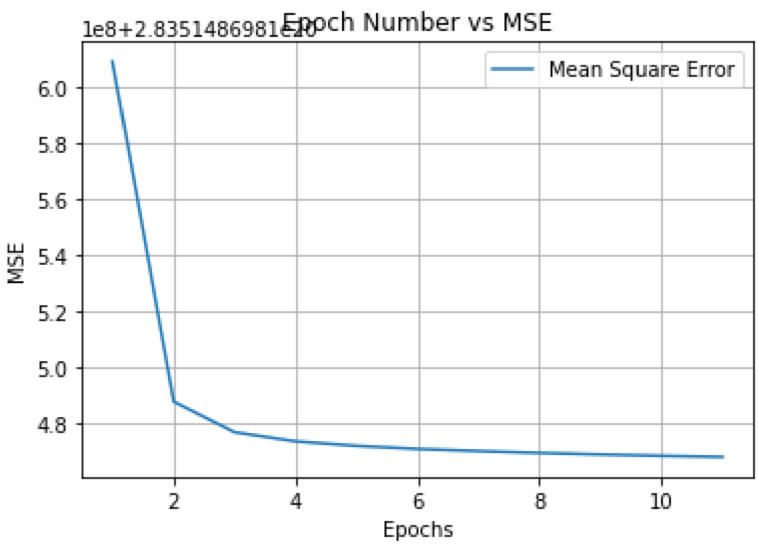
```
    Epoch Number =  1  The mean square error is =   2.8351486980960918e+20
    Epoch Number =  2  The mean square error is =   2.8351486980948748e+20
    Epoch Number =  3  The mean square error is =   2.835148698094765e+20
    Epoch Number =  4  The mean square error is =   2.8351486980947326e+20
    Epoch Number =  5  The mean square error is =   2.8351486980947162e+20
    Epoch Number =  6  The mean square error is =   2.8351486980947054e+20
    Epoch Number =  7  The mean square error is =   2.8351486980946975e+20
    Epoch Number =  8  The mean square error is =   2.835148698094691e+20
    Epoch Number =  9  The mean square error is =   2.8351486980946857e+20
    Epoch Number =  10  The mean square error is =   2.8351486980946808e+20
    Epoch Number =  11  The mean square error is =   2.8351486980946765e+20
```

Plot epoch number vs MSE

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

```
s = [2.8351486980960918e+20, 2.8351486980948748e+20, 2.835148698094765e+20, 2.835148698094
h = [1,2,3,4,5,6,7,8,9,10,11]
```

```
plt.plot(h, s, label="Mean Square Error")
plt.grid()
plt.title('Epoch Number vs MSE')
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.legend()
plt.show()
```



Saved successfully!

- 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.

```
from google.colab import files
files= files.upload()
```

Choose Files user_info.csv

- **user_info.csv**(application/vnd.ms-excel) - 13017 bytes, last modified: 12/1/2021 - 100% done

```
import pandas as pd
data1=pd.read_csv('user_info.csv')
data1.head()
```

	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

```
data1.describe()
```

	user_id	age	is_male	orig_user_id
count	943.000000	943.000000	943.000000	943.000000
mean	471.000000	34.051962	0.710498	472.000000
std	272.364951	12.192740	0.453772	272.364951
min	0.000000	7.000000	0.000000	1.000000
[illegible]	[illegible]	[illegible]	0.000000	236.500000
[illegible]	[illegible]	[illegible]	1.000000	472.000000
75%	706.500000	43.000000	1.000000	707.500000
max	942.000000	73.000000	1.000000	943.000000

Saved successfully!

```
data1.shape
```

```
(943, 4)
```

```
# remove the 'is_male' feature
data1.drop(['is_male'], axis=1)
y_true = data1['is_male']
data1.drop(['is_male'], axis=1, inplace=True)
```

```
data1.head()
```

	user_id	age	orig_user_id
0	0	24	1
1	1	53	2
4	4	33	5

```
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(data1, y_true, stratify=y_true, test_si
```

```
print("Number of data points in train data :",X_train.shape)
print("Number of data points in test data :",X_test.shape)
```

```
Number of data points in train data : (660, 3)
Number of data points in test data : (283, 3)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine # database connection
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
uncatedSVD
rmalize
import CountVectorizer
from sklearn.manifold import TSNE
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix

#from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.feature_extraction.text import TfidfVectorizer
from collections import Counter
from scipy.sparse import hstack
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
#from sklearn.cross_validation import StratifiedKFold
from collections import Counter, defaultdict
from sklearn.calibration import CalibratedClassifierCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import math
from sklearn.metrics import normalized_mutual_info_score
from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import cross_val_score
```

```
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import SGDClassifier
#from mlxtend.classifier import StackingClassifier

from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve, auc, roc_curve
```

```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train)
train_len = len(y_train)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/train_len
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test)
test_len = len(y_test)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.28939393939393937 Class 1:  0.7106060606060606
---------- Distribution of output variable in train data ----------
Class 0:  0.7102473498233216 Class 1:  0.7102473498233216
```

```
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted

Saved successfully!
                   sion matrix with the sum of elements in that column

    # C = [[1, 2],
    #     [3, 4]]
    # C.T = [[1, 3],
    #        [2, 4]]
    # C.sum(axis = 1)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
    # C.sum(axix =1) = [[3, 7]]
    # ((C.T)/(C.sum(axis=1))) = [[1/3, 3/7]
    #                           [2/3, 4/7]]

    # ((C.T)/(C.sum(axis=1))).T = [[1/3, 2/3]
    #                           [3/7, 4/7]]
    # sum of row elements = 1

    B =(C/C.sum(axis=0))
    #divid each element of the confusion matrix with the sum of elements in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows in two
    # C.sum(axix =0) = [[4, 6]]
    # (C/C.sum(axis=0)) = [[1/4, 2/6],
    #                      [3/4, 4/6]]
    plt.figure(figsize=(20,4))
```

```
    labels = [1,2]
    # representing A in heatmap format
    cmap=sns.light_palette("blue")
    plt.subplot(1, 3, 1)
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=label
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

    plt.subplot(1, 3, 2)
    sns.heatmap(B, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=label
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Precision matrix")

    plt.subplot(1, 3, 3)
    # representing B in heatmap format
    sns.heatmap(A, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=label
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

```
from sklearn.metrics import log_loss

# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to genarate 9 numbers and divide each of the numbers by their sum
# ref: https://stackoverflow.com/a/18662466/4084039
exactly same size as the CV data

for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])
print("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y, eps=1e-15))

predicted_y =np.argmax(predicted_y, axis=1)
plot_confusion_matrix(y_test, predicted_y)
```

Saved successfully!

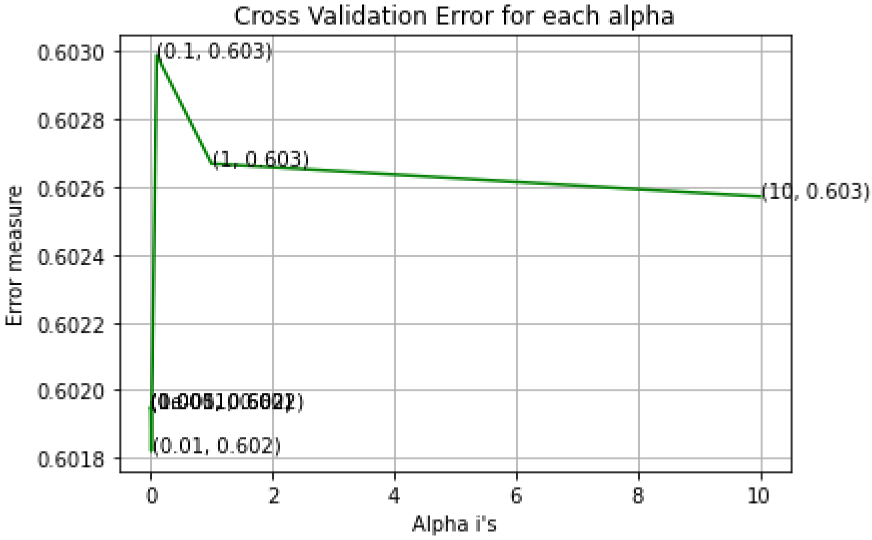
```
Log loss on Test Data using Random Model 0.9833389311060641
```

```
alpha = [10 ** x for x in range(-5, 2)] # hyperparameter for SGD classifier.

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/generated/skle
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=Tru
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optima
# class_weight=None, warm_start=False, average=False, n_iter=None)

# some of methods
# fit(X, y[, coef_init, intercept_init, …]) Fit linear model with Stochastic Gradient Desc
# predict(X)  Predict class labels for samples in X.

#------------------------------
# video link:
#------------------------------



log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labe
```

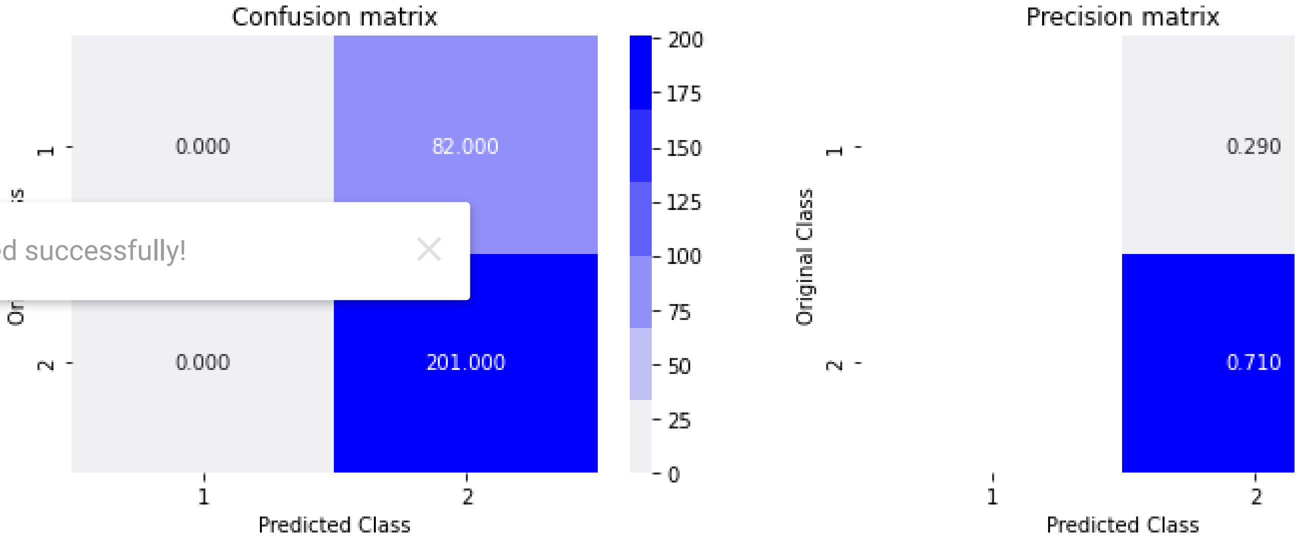
Saved successfully!

```
)
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
clf.fit(X_train, y_train)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_train, y_train)

predict_y = sig_clf.predict_proba(X_train)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(
predict_y = sig_clf.predict_proba(X_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6019488299906202
For values of alpha =  0.0001 The log loss is: 0.6019488299906202
For values of alpha =  0.001 The log loss is: 0.6019488299906202
For values of alpha =  0.01 The log loss is: 0.6018202107993741
For values of alpha =  0.1 The log loss is: 0.6029886416788311
For values of alpha =  1 The log loss is: 0.6026688438473562
For values of alpha =  10 The log loss is: 0.6025716471318935
```




```
For values of best alpha =  0.01 The train log loss is: 0.6020374280079639
For values of best alpha =  0.01 The test log loss is: 0.6018202107993741
Total number of data points : 283
```



Saved successfully!

Thanks Applied AI, it was a really nice assignment.

1s completed at 12:58 AM

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

Saved successfully!