In [2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
import seaborn as sns

pd.set_option('display.max_columns', None)
path = 'datasets/'
```

In [9]:
```python
data = pd.read_csv(path + '4-train_final_96features.csv')
data.head()
```

Out[9]:

	id	is_duplicate	freq_qid1_x	freq_qid2_x	q1_len_x	q2_len_x	num_words_q1_x	num_words_q2_x	common_word_q12_x	total_word_q12_x	shared_words_q12_x
0	0	0	1	1	66	57	14	12	10	23	0.43478
1	1	0	4	1	51	88	8	13	4	20	0.20000
2	2	0	1	1	73	59	14	10	4	24	0.16666
3	3	0	1	1	50	65	11	9	0	19	0.00000
4	4	0	3	1	76	39	13	7	2	20	0.10000

In [10]:
```python
data.shape
```

Out[10]:
```
(404290, 231)
```

Dividing Data

In [11]:
```python
X = data.drop(['id','is_duplicate'],axis=1)
y = data['is_duplicate']
```

In [12]:
```python
X.shape, y.shape
```

Out[12]:
```
((404290, 229), (404290,))
```

Train Test Split-[80:20]

In [14]:
```python
from sklearn.model_selection import train_test_split
```

In [38]:
```python
#train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2,random_state=108)
```

In [39]:
```python
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[39]:
```
((323432, 229), (80858, 229), (323432,), (80858,))
```

In [40]:
```python
st = 'Data points in '
print(f'{st} [train data] ==> {X_train.shape}')
print(f'{st} [test data] ==> {X_test.shape}')
```

```
Data points in  [train data] ==> (323432, 229)
Data points in  [test data] ==> (80858, 229)
```

`Observation:`

- Number of data points in train data : (323432, 229)
- Number of data points in test data : (80858, 229)

Counters Link

In [41]:
```python
from collections import Counter
```

In [42]:
```python
print('Distribution of output variable in [ train data ]\n','***'*10)

train_distr = Counter(y_train)
train_len = len(y_train)

print(f'Class-0: {int(train_distr[0])/train_len}\nClass-1: {int(train_distr[1])/train_len}')
```

```
Distribution of output variable in [ train data ]
 ******************************
Class-0: 0.6308033837097133
Class-1: 0.36919661629028666
```

In [43]:
```python
print('Distribution of output variable in [ test data ]\n','***'*10)

test_distr = Counter(y_test)
test_len = len(y_test)

print(f'Class-0: {int(test_distr[1])/test_len}\nClass-1: {int(train_distr[1])/train_len}')
```

```
Distribution of output variable in [ test data ]
 ******************************
Class-0: 0.36920279997031835
Class-1: 0.36919661629028666
```

Ploting Confusion Matrices

- `C` = `9,9 matrix`, each cell (i,j) represents number of points of class i are predicted class j

```
                              Calculating for [ A ]

   divid each element of the confusion matrix with 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 diamensional array
   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

                            Calculating for [ B ]

 divid each element of confusion matrix with 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 diamensional array
  C.sum(axix =0) = [[4, 6]]
  (C/C.sum(axis=0)) = [[1/4, 2/6],
                       [3/4, 4/6]]
```

In [45]:
```python
#function plots confusion matrices given y_i, y_i_hat
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    A = (((C.T)/(C.sum(axis=1))).T)
    B =(C/C.sum(axis=0))
    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=labels)
    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=labels)
    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=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")
    plt.savefig('output_img/10-xgboost_confusionMatrix.png') #saving image output
    plt.show()
```

Random Model

`Finding worst-case log-loss`

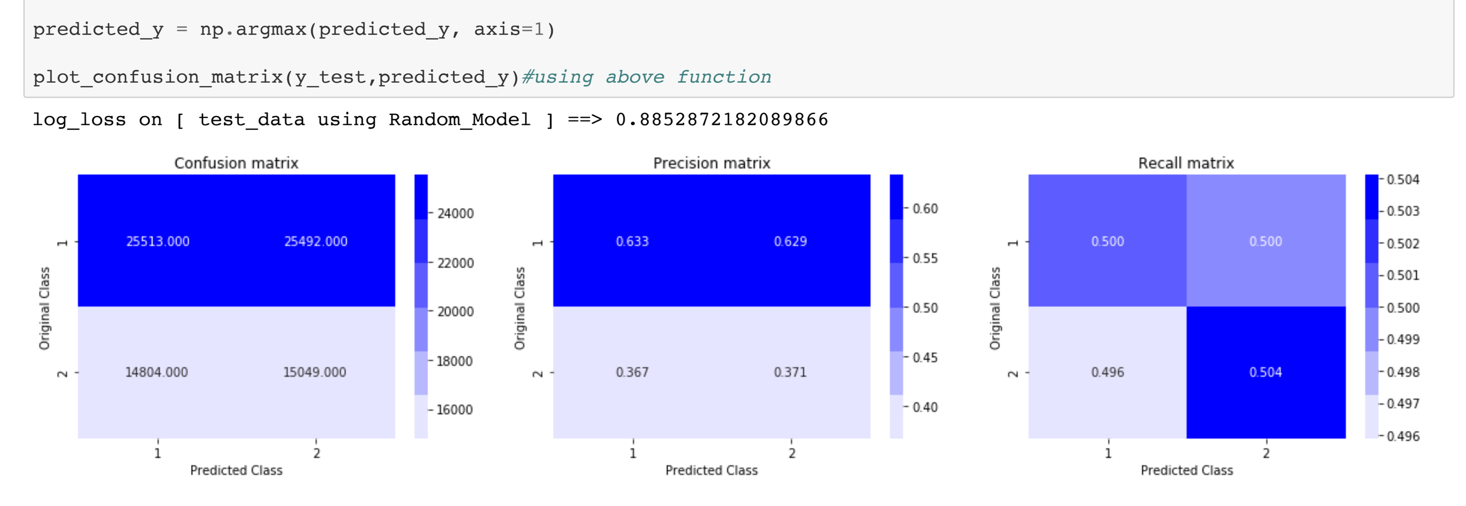
- will generate 9 numbers and sum of numbers should be 1
- one solution is to genarate 9 numbers and divide each of numbers by their sum
- will create a output array that has exactly same size as the CV data

Generating a list of random numbers summing to 1

In [48]:
```python
from sklearn.metrics import log_loss
from sklearn.metrics import confusion_matrix
```

In [55]:
```python
predicted_y = np.zeros((test_len,2))
for i in range(test_len):
    rand_probs = np.random.rand(1,2)
    predicted_y[i] = ((rand_probs/sum(sum(rand_probs)))[0])

print(f'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)#using above function
```

```
log_loss on [ test_data using Random_Model ] ==> 0.8852872182089866
```



SGDClassifier & Hyperparameter Tuning

In [56]:
```python
#hyperparam
alpha = [10 ** x for x in range(-5,2)]
alpha
```

Out[56]:
```
[1e-05, 0.0001, 0.001, 0.01, 0.1, 1, 10]
```

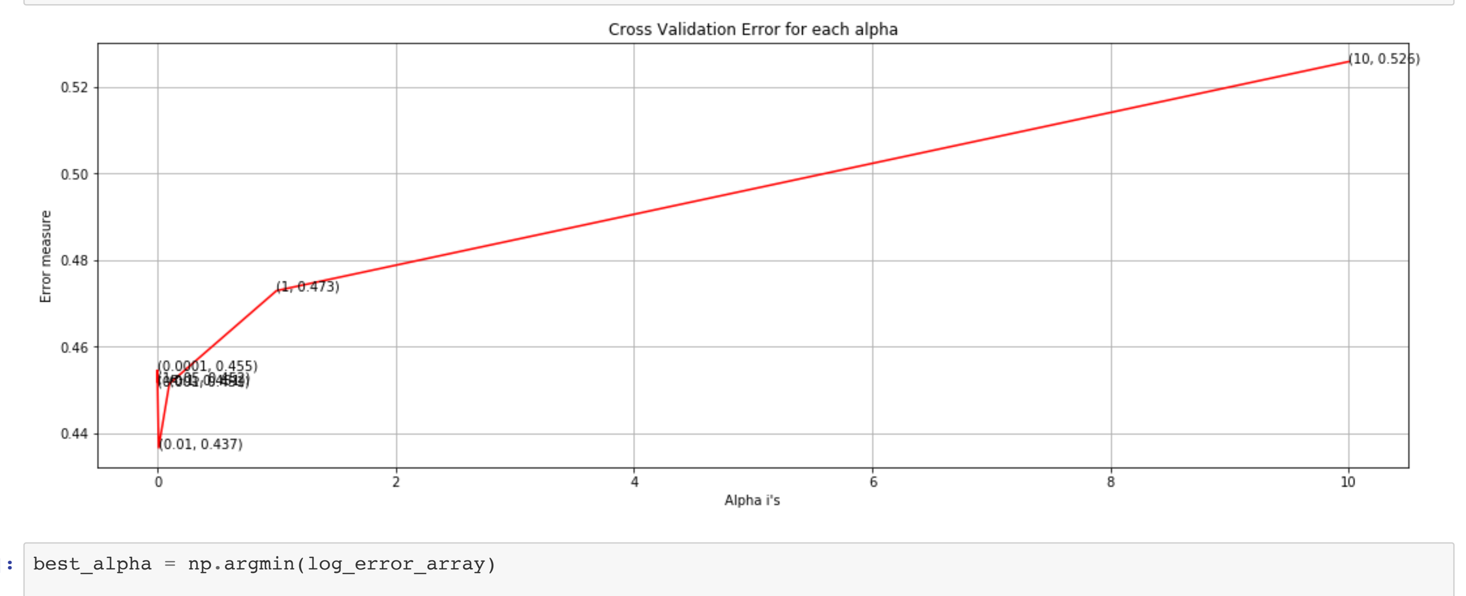
SGD-Classifier Docs

In [60]:
```python
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
```

In [61]:
```python
alpha = [10 ** x for x in range(-5,2)]
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i,penalty='l2',loss='log',random_state=108)
    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(f'For values of alpha = {i}, log loss is: {log_loss(y_test,predict_y,labels=clf.classes_,eps=1e-15)}')
```

```
For values of alpha = 1e-05, log loss is: 0.4518226263478587
For values of alpha = 0.0001, log loss is: 0.45451812253622154
For values of alpha = 0.001, log loss is: 0.45102033471070324
For values of alpha = 0.01, log loss is: 0.43661784628300165
For values of alpha = 0.1, log loss is: 0.45141877660864926
For values of alpha = 1, log loss is: 0.4728840456072604
For values of alpha = 10, log loss is: 0.5257608247242521
```

In [87]:
```python
fig,ax = plt.subplots(nrows=1,ncols=1,figsize=(18,6))
ax.plot(alpha,log_error_array,c='r')
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]))

#ploting grid in backend
plt.grid()
plt.title('Cross Validation Error for each alpha')
plt.xlabel("Alpha i's")
plt.ylabel('Error measure')
plt.show()
```

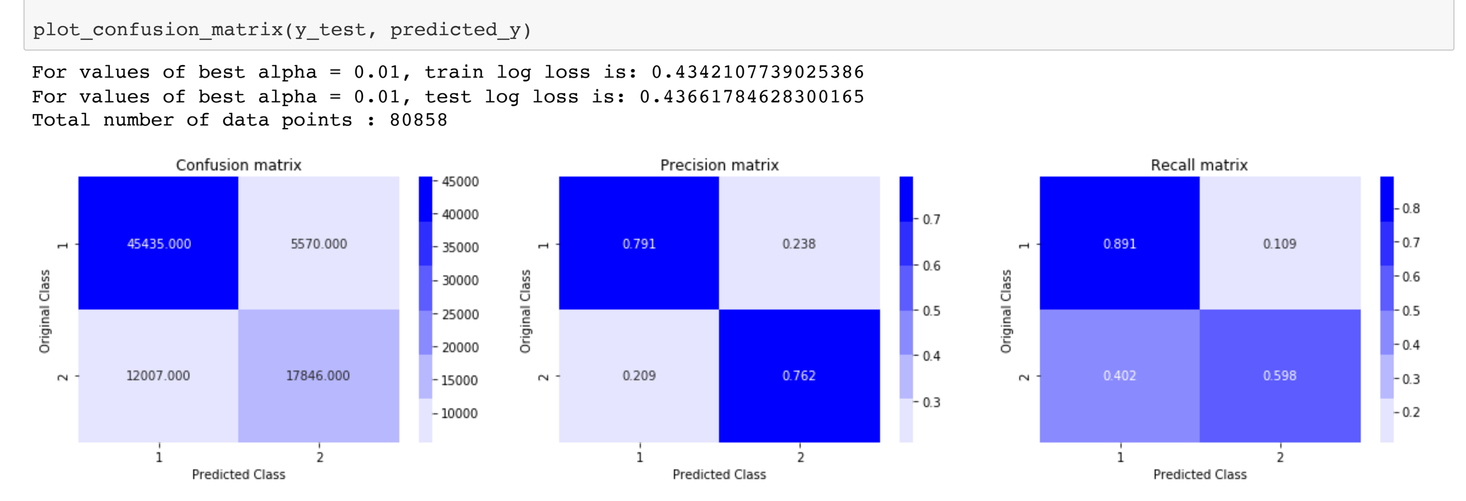


In [63]:
```python
best_alpha = np.argmin(log_error_array)

clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=108)
clf.fit(X_train,y_train)

sig_clf = CalibratedClassifierCV(clf, method='sigmoid')
sig_clf.fit(X_train, y_train)
```

Out[63]:
```
CalibratedClassifierCV(base_estimator=SGDClassifier(alpha=0.01, loss='log',
                                                    random_state=108))
```

In [64]:
```python
predict_y = sig_clf.predict_proba(X_train)
print(f'For values of best alpha = {alpha[best_alpha]}, train log loss is: {log_loss(y_train, predict_y, labels=clf.cl
asses_, eps=1e-15)}')

predict_y = sig_clf.predict_proba(X_test)
print(f'For values of best alpha = {alpha[best_alpha]}, test log loss is: {log_loss(y_test, predict_y, labels=clf.clas
ses_, eps=1e-15)}')

predicted_y =np.argmax(predict_y,axis=1)
print(f'Total number of data points : {len(predicted_y)}')

plot_confusion_matrix(y_test, predicted_y)
```

```
For values of best alpha = 0.01, train log loss is: 0.4342107739025386
For values of best alpha = 0.01, test log loss is: 0.43661784628300165
Total number of data points : 80858
```



XGBoost & Hyperparameter Tuning

In [65]:
```python
import xgboost as xgb
```

In [66]:
```python
#global
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['eta'] = 0.02
params['max_depth'] = 4
```

In [67]:
```python
X_train.columns.tolist()
type(X_train)
```

Out[67]:
```
pandas.core.frame.DataFrame
```

In [68]:
```python
X_test.columns.tolist()
type(X_test)
```

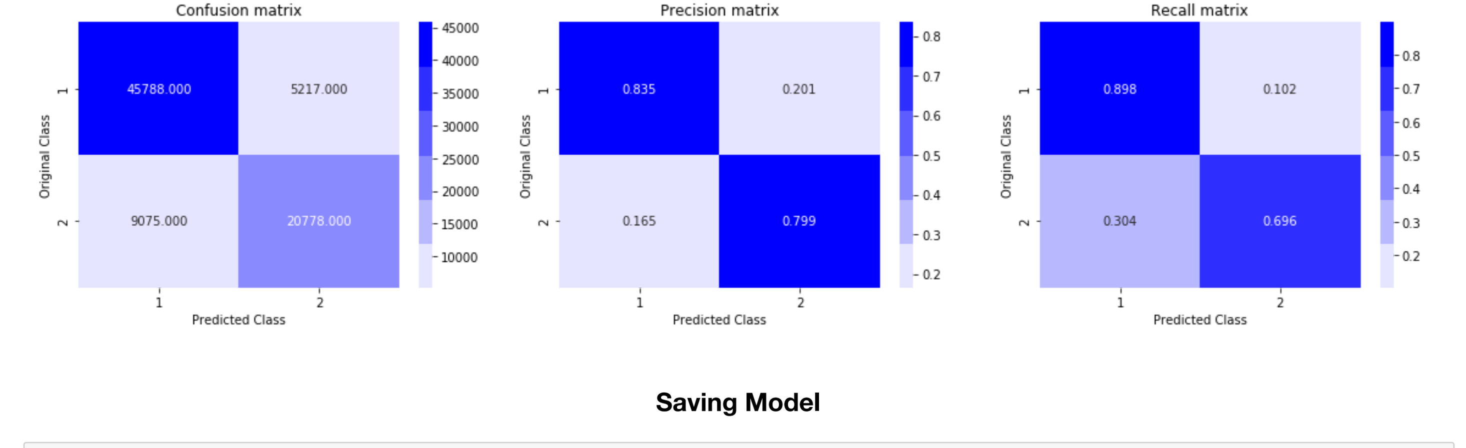
Out[68]:
```
pandas.core.frame.DataFrame
```

In [69]:
```python
d_train = xgb.DMatrix(X_train, label=y_train)
d_test = xgb.DMatrix(X_test, label=y_test)
```

In [70]:
```python
watchlist = [(d_train, 'train'), (d_test, 'valid')]
```

In [71]:
```python
bst = xgb.train(params, d_train, 400, watchlist, early_stopping_rounds=20, verbose_eval=10)

xgdmat = xgb.DMatrix(X_train,y_train)
predict_y = bst.predict(d_test)

print(f'test log loss is: {log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15)}')

predicted_y = np.array(predict_y > 0.5,dtype=int)
print(f'total number of data points : {len(predicted_y)}')

plot_confusion_matrix(y_test, predicted_y)
```

```
[0]	train-logloss:0.68461	valid-logloss:0.68474
Multiple eval metrics have been passed: 'valid-logloss' will be used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]	train-logloss:0.61512	valid-logloss:0.61529
[20]	train-logloss:0.56448	valid-logloss:0.56484
[30]	train-logloss:0.52630	valid-logloss:0.52678
[40]	train-logloss:0.49702	valid-logloss:0.49746
[50]	train-logloss:0.47420	valid-logloss:0.47459
[60]	train-logloss:0.45585	valid-logloss:0.45632
[70]	train-logloss:0.44135	valid-logloss:0.44186
[80]	train-logloss:0.42944	valid-logloss:0.42996
[90]	train-logloss:0.41994	valid-logloss:0.42047
[100]	train-logloss:0.41209	valid-logloss:0.41265
[110]	train-logloss:0.40543	valid-logloss:0.40599
[120]	train-logloss:0.39988	valid-logloss:0.40039
[130]	train-logloss:0.39501	valid-logloss:0.39555
[140]	train-logloss:0.39109	valid-logloss:0.39163
[150]	train-logloss:0.38751	valid-logloss:0.38809
[160]	train-logloss:0.38459	valid-logloss:0.38516
[170]	train-logloss:0.38189	valid-logloss:0.38249
[180]	train-logloss:0.37960	valid-logloss:0.38026
[190]	train-logloss:0.37757	valid-logloss:0.37822
[200]	train-logloss:0.37583	valid-logloss:0.37650
[210]	train-logloss:0.37412	valid-logloss:0.37484
[220]	train-logloss:0.37251	valid-logloss:0.37328
[230]	train-logloss:0.37102	valid-logloss:0.37179
[240]	train-logloss:0.36958	valid-logloss:0.37048
[250]	train-logloss:0.36807	valid-logloss:0.36896
[260]	train-logloss:0.36683	valid-logloss:0.36776
[270]	train-logloss:0.36554	valid-logloss:0.36658
[280]	train-logloss:0.36432	valid-logloss:0.36543
[290]	train-logloss:0.36324	valid-logloss:0.36440
[300]	train-logloss:0.36212	valid-logloss:0.36333
[310]	train-logloss:0.36106	valid-logloss:0.36233
[320]	train-logloss:0.36012	valid-logloss:0.36147
[330]	train-logloss:0.35914	valid-logloss:0.36057
[340]	train-logloss:0.35824	valid-logloss:0.35972
[350]	train-logloss:0.35738	valid-logloss:0.35894
[360]	train-logloss:0.35652	valid-logloss:0.35816
[370]	train-logloss:0.35572	valid-logloss:0.35740
[380]	train-logloss:0.35496	valid-logloss:0.35673
[390]	train-logloss:0.35422	valid-logloss:0.35608
[399]	train-logloss:0.35360	valid-logloss:0.35552
test log loss is: 0.35552283332383183
total number of data points : 80858
```



Saving Model

In [72]:
```python
import pickle
```

In [73]:
```python
filename = 'models/xgboost_96.sav'
pickle.dump(bst, open(filename,'wb')) #write binary mode
```

In [74]:
```python
#loading model from disk
loaded_model = pickle.load(open(filename,'rb')) #read binary mode
print(loaded_model)
```

```
<xgboost.core.Booster object at 0x149c2fdd0>
```

In [ ]: