Final ML Modeling

Now we create ML model for our final data(preprocessed and featurized).

In [2]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import sqlite3
from sqlalchemy import create_engine
import csv
import os
warnings.filterwarnings("ignore")
import datetime as dt
import numpy as np
from nltk.corpus import stopwords
from sklearn.decomposition import TruncatedSVD
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text 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.model_selection 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.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
```

In [3]:
```
# mounting google drive
from google.colab import drive
drive.mount('/content/drive/')
```

```
Mounted at /content/drive/
```

In [4]:
```
data = pd.read_csv('/content/drive/My Drive/ML_Projects/final_data.csv')
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Columns: 221 entries, Unnamed: 0 to 95_y
dtypes: float64(209), int64(10), object(2)
memory usage: 681.7+ MB
```

In [ ]:

```
data.head()
```

Out[ ]:

	Unnamed: 0	id	qid1	qid2	question1	question2	is_duplicate	q1_len	q2_len	q1_words
0	0	0	1	2	step step guide invest share market india	step step guide invest share market	0	41.0	35.0	6.0
1	1	1	3	4	story kohinoor koh noor diamond	would happen indian government stole kohinoor ...	0	31.0	67.0	5.0
2	2	2	5	6	increase speed internet connection using vpn	internet speed increased hacking dns	0	44.0	36.0	6.0
3	3	3	7	8	mentally lonely solve	find remainder math 23 24 math divided 24 23	0	21.0	44.0	3.0
4	4	4	9	10	one dissolve water quikly sugar salt methane c...	fish would survive salt water	0	60.0	29.0	10.0

5 rows × 221 columns

In [5]:

```
y_true = data['is_duplicate']
data.drop(['Unnamed: 0', 'id','qid1', 'qid2', 'question1', 'question2' ,'is_d
```

In [ ]:

```
data.head()
```

Out[ ]:

	qid1	qid2	question1	question2	q1_len	q2_len	q1_words	q2_words	words_total	wor
0	1	2	step step guide invest share market india	step step guide invest share market	41.0	35.0	6.0	5.0	11.0	

	qid1	qid2	question1	question2	q1_len	q2_len	q1_words	q2_words	words_total	wor
1	3	4	story kohinoor koh noor diamond	would happen indian government stole kohinoor ...	31.0	67.0	5.0	10.0	15.0	
2	5	6	increase speed internet connection using vpn	internet speed increased hacking dns	44.0	36.0	6.0	5.0	11.0	
3	7	8	mentally lonely solve	find remainder math 23 24 math divided 24 23	21.0	44.0	3.0	6.0	9.0	
4	9	10	one dissolve water quikly sugar salt methane c...	fish would survive salt water	60.0	29.0	10.0	5.0	15.0	

5 rows × 218 columns

Train - Test Split(70 : 30)

In [ ]:

```

```

Out[ ]:

```
pandas.core.series.Series
```

In [6]:

```
x_train, x_test, y_train, y_test = train_test_split(data, y_true, test_size =
```

In [ ]:

```
print("Number of data points in train : ", x_train.shape[0])
print("Number of data point in test : ", x_test.shape[0])
```

```
Number of data points in train :  283003
Number of data point in test :  121287
```

In [7]:

```
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

    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)
```

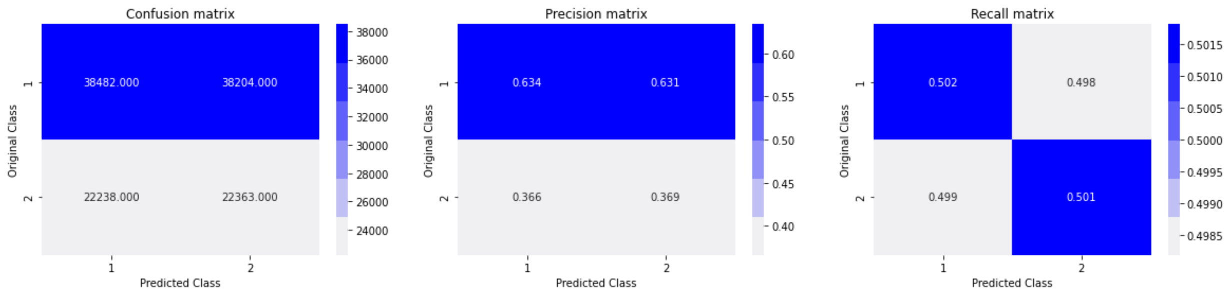
```
    sns.heatmap(C, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, ytic
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Confusion matrix")

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

    plt.show()
```

2.1 Building Random Model for worst case scnerio

In [ ]:

```
#exactly same size as the CV data
test_len = len(y_test)
train_len = len(y_train)
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("Log loss on Test Data using Random Model",log_loss(y_test, predicted_y

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

```
Log loss on Test Data using Random Model 0.8843309033276239
```



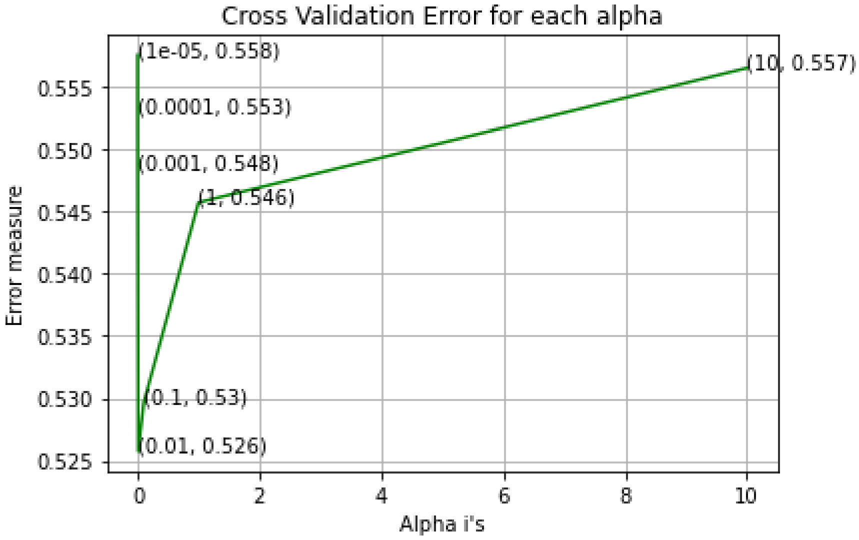
2.2 Logistic Regression with Hyperparameter tuning

In [ ]:

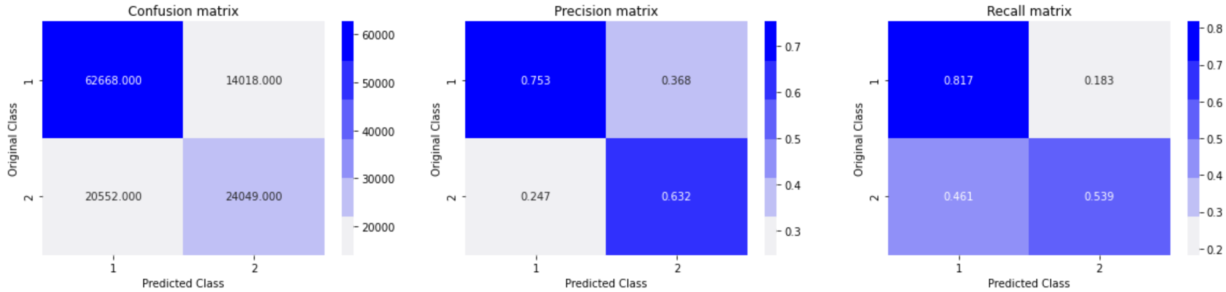
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    clf.fit(x_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train, y_train)
    predict_y = sig_clf.predict_proba(x_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, e
```

```
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pr

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
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_
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 i
predict_y = sig_clf.predict_proba(x_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
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: 0.5576142951160223
For values of alpha =  0.0001 The log loss is: 0.5530489946987308
For values of alpha =  0.001 The log loss is: 0.548445229824679
For values of alpha =  0.01 The log loss is: 0.5257417560363247
For values of alpha =  0.1 The log loss is: 0.529612794691027
For values of alpha =  1 The log loss is: 0.5457715401514471
For values of alpha =  10 The log loss is: 0.5565646255826227
```




```
For values of best alpha =  0.01 The train log loss is: 0.5228870308221469
For values of best alpha =  0.01 The test log loss is: 0.5257417560363247
Total number of data points : 121287
```

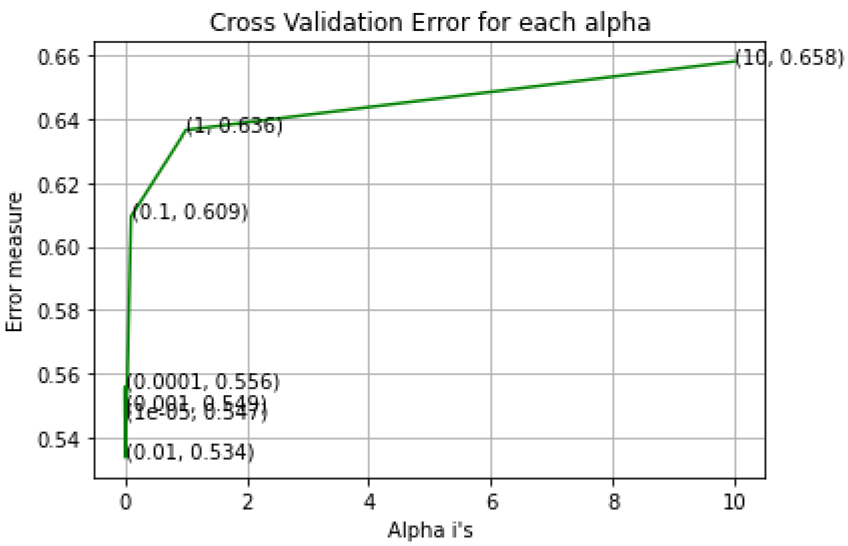


2.3 Linear SVM with hyperparamter tuning

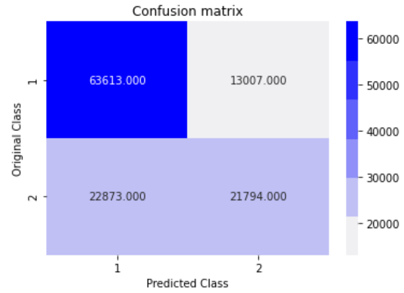
In [8]:

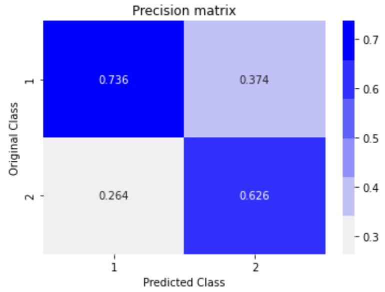
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', 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_, e
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pr

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
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='l1', loss='hinge', rand
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 i
predict_y = sig_clf.predict_proba(x_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

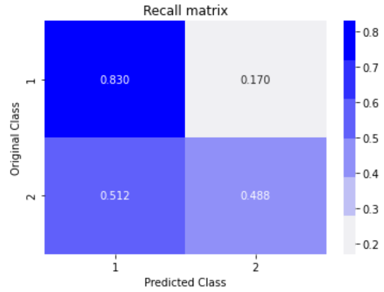
```
For values of alpha =  1e-05 The log loss is: 0.5465807687373918
For values of alpha =  0.0001 The log loss is: 0.5558197431968697
For values of alpha =  0.001 The log loss is: 0.5491315497596453
For values of alpha =  0.01 The log loss is: 0.533789863057763
For values of alpha =  0.1 The log loss is: 0.6093873763222306
For values of alpha =  1 The log loss is: 0.6364525880912241
For values of alpha =  10 The log loss is: 0.6580351094741663
```



```
For values of best alpha =  0.01 The train log loss is: 0.5338106610798011
For values of best alpha =  0.01 The test log loss is: 0.533789863057763
Total number of data points : 121287
```







Conclusion

So we have our Results now for both the model.

1. Logistic Regression with Hyperparameter tuning : best aplha = 0.01 with log-loss = 0.52 (approx)
2. Linear SVM with Hyperparameter tuning : best alpha = 0.01 with log-loss = 0.53

Hence, both the models performs similar. But logistic Regression is fast for larger datasets hence, we can consider logistic regression as a best fit among these two models.