In [1]:

```
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
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 [67]:

```
# 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

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in t

    # 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
    # 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 t
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to
    # 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, 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()
```

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.

Preparing data

In [3]:

```
# avoid decoding problems
df = pd.read_csv("train.csv", nrows=100000)

df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))

df.head()
```

Out[3]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

In [4]:

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('nlp_features_train.csv'):
    dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
else:
    print("download nlp_features_train.csv from drive or run previous noteboo

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='la
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run p
```

In [5]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis
df1.head(2)
```

Out[5]:

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	

In [6]:

```
df2.head(2)
```

Out[6]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Co
0	0	1	1	66	57	14	12	
1	1	4	1	51	88	8	13	

In [7]:

```
df1.head(2)
```

Out[7]:

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	

In [8]:

```
df2.head(2)
```

Out[8]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Co
0	0	1	1	66	57	14	12	
1	1	4	1	51	88	8	13	

In [9]:

```
df = df.drop(['qid1','qid2'],axis=1)
df.head(2)
```

Out[9]:

	id	question1	question2	is_duplicate
0	0	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0

In [10]:

```
f1  = df1
f2  = f1.merge(df2, on='id',how='left')
X = f2.merge(df, on='id',how='left')
X.head(2)
```

Out[10]:

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	

2 rows × 30 columns

In [11]:

```
X.shape
```

Out[11]:

```
(404290, 30)
```

In [12]:

```
y_true = X['is_duplicate']
X.drop(['id','is_duplicate'], axis=1, inplace=True)

X.head()
```

Out[12]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0

5 rows × 28 columns

In [13]:

```
print(X.shape, len(y_true))
```

```
(404290, 28) 404290
```

In [14]:

```
X_train, X_test, y_train, y_test = train_test_split(X.head(100000), y_true[0:

print(X_train.shape,len(y_train))
print(X_test.shape, len(y_test))
```

```
(70000, 28) 70000
(30000, 28) 30000
```

In [15]:

```
X_train.head(2)
```

Out[15]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_worc
20876	0.399992	0.333328	0.666644	0.399992	0.499994	0.363633	
83126	0.666644	0.499988	0.999967	0.599988	0.833319	0.499995	

2 rows × 28 columns

In [16]:

```
X_test.head(2)
```

Out[16]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_worc
30624	0.000000	0.000000	0.333322	0.199996	0.142855	0.099999	
20821	0.999967	0.749981	0.999975	0.999975	0.874989	0.874989	

2 rows × 28 columns

Vectorizing Question 1

In [17]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer_tfidf_q1 = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_featu
vectorizer_tfidf_q1.fit(X_train['question1'].values)

X_train_q1_tfidf = vectorizer_tfidf_q1.transform(X_train['question1'].values)
X_test_q1_tfidf = vectorizer_tfidf_q1.transform(X_test['question1'].values)

print("After vectorizations")
print(X_train_q1_tfidf.shape, y_train.shape)
print(X_test_q1_tfidf.shape, y_test.shape)
```

```
After vectorizations
(70000, 5000) (70000,)
(30000, 5000) (30000,)
```

Vectorizing Question 2

In [18]:

```
vectorizer_tfidf_q2 = TfidfVectorizer(min_df=10, ngram_range=(1,4), max_featu
vectorizer_tfidf_q2.fit(X_train['question2'].values)

X_train_q2_tfidf = vectorizer_tfidf_q2.transform(X_train['question2'].values)
X_test_q2_tfidf = vectorizer_tfidf_q2.transform(X_test['question2'].values)

print("After vectorizations")
print(X_train_q2_tfidf.shape, y_train.shape)
print(X_test_q2_tfidf.shape, y_test.shape)
```

```
After vectorizations
(70000, 5000) (70000,)
(30000, 5000) (30000,)
```

In [19]:

```
q1_train = pd.DataFrame(X_train_q1_tfidf.toarray())
q2_train = pd.DataFrame(X_train_q2_tfidf.toarray())
q1_test = pd.DataFrame(X_test_q1_tfidf.toarray())
q2_test = pd.DataFrame(X_test_q2_tfidf.toarray())
```

In [20]:

```
X_train.drop(['question1','question2'], axis=1, inplace=True)
X_train.head(2)
```

Out[20]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_worc
20876	0.399992	0.333328	0.666644	0.399992	0.499994	0.363633	
83126	0.666644	0.499988	0.999967	0.599988	0.833319	0.499995	

2 rows × 26 columns

In [21]:

```
X_train = pd.concat([X_train.reset_index(drop=True),q1_train.reset_index(drop
X_train.shape
```

Out[21]:

```
(70000, 10026)
```

X_train with obtained TFIDF features for Question 1 and Question 2

(26 + 5000 + 5000)

In [22]:

```
X_train.head(2)
```

Out[22]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.399992	0.333328	0.666644	0.399992	0.499994	0.363633	1.0
1	0.666644	0.499988	0.999967	0.599988	0.833319	0.499995	0.0

2 rows × 10026 columns

In [23]:

```
X_test.drop(['question1','question2'], axis=1, inplace=True)
X_test.head(2)
```

Out[23]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_worc
30624	0.000000	0.000000	0.333322	0.199996	0.142855	0.099999	
20821	0.999967	0.749981	0.999975	0.999975	0.874989	0.874989	

2 rows × 26 columns

In [24]:

```
X_test = pd.concat([X_test.reset_index(drop=True),q1_test.reset_index(drop=Tı
X_test.shape
```

Out[24]:

```
(30000, 10026)
```

X_test with obtained TFIDF features for Question 1 and Question 2 (26 + 5000 + 5000)

In [25]:

```
X_test.head(2)
```

Out[25]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.000000	0.000000	0.333322	0.199996	0.142855	0.099999	0.0
1	0.999967	0.749981	0.999975	0.999975	0.874989	0.874989	0.0

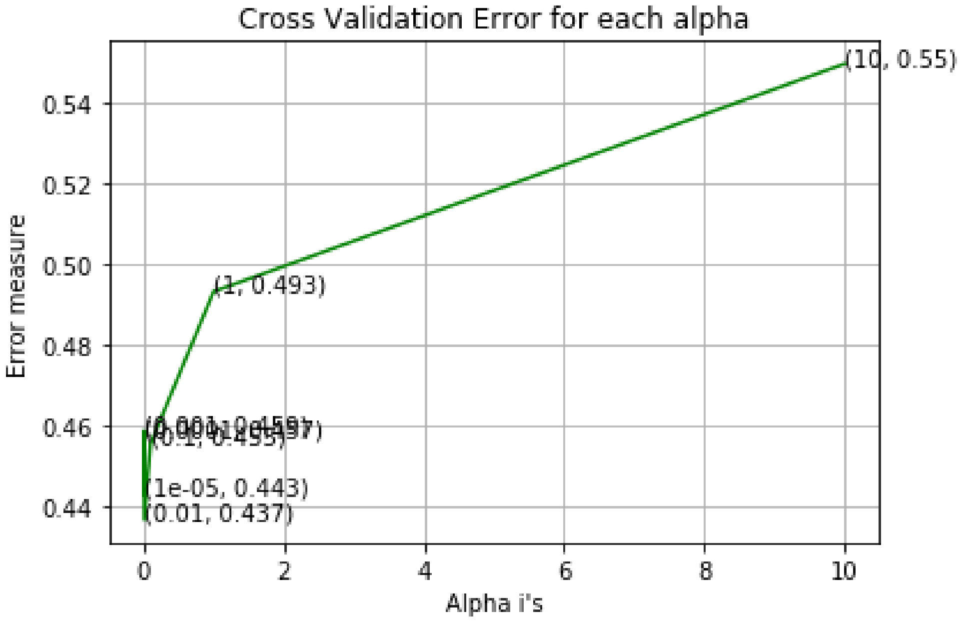
2 rows × 10026 columns

Logistic regression

In [29]:

```
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', randor
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))
```

```
For values of alpha =  1e-05 The log loss is: 0.44297644978307
76
For values of alpha =  0.0001 The log loss is: 0.4574031536123
3175
For values of alpha =  0.001 The log loss is: 0.45872881579341
734
For values of alpha =  0.01 The log loss is: 0.436689240300541
7
For values of alpha =  0.1 The log loss is: 0.4554943357379192
For values of alpha =  1 The log loss is: 0.4931462750304008
For values of alpha =  10 The log loss is: 0.5495396066805692
```



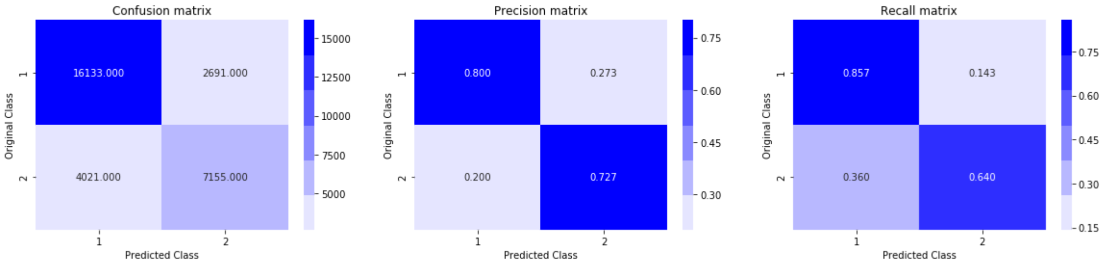

```
For values of best alpha =  0.01 The train log loss is: 0.4308
9543944564196
For values of best alpha =  0.01 The test log loss is: 0.43668
92403005417
Total number of data points : 30000

---------------------------------------------------------------
-------------
NameError                                 Traceback (most rece
nt call last)
<ipython-input-29-54cba29181c3> in <module>
     50 predicted_y =np.argmax(predict_y,axis=1)
     51 print("Total number of data points :", len(predicted_y
))
---> 52 plot_confusion_matrix(y_test, predicted_y)

NameError: name 'plot_confusion_matrix' is not defined
```

Reason for the above error

- The above error occured because, I forgot to run plot_confusion_matrix function
- Resolved by running the cell containing plot_confusion_matrix function

In [31]:

```
plot_confusion_matrix(y_test, predicted_y)
```

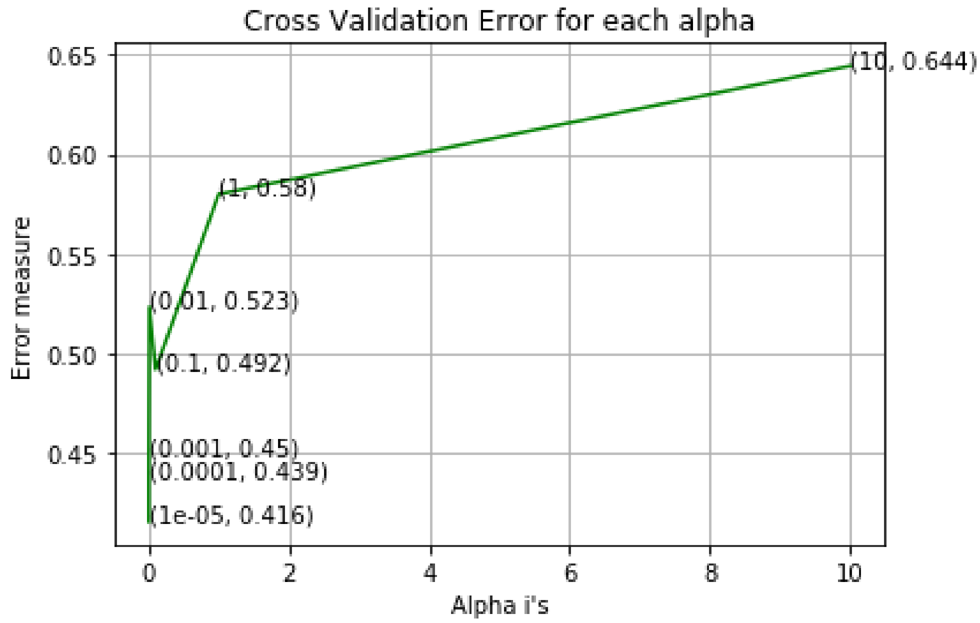


Linear SVM

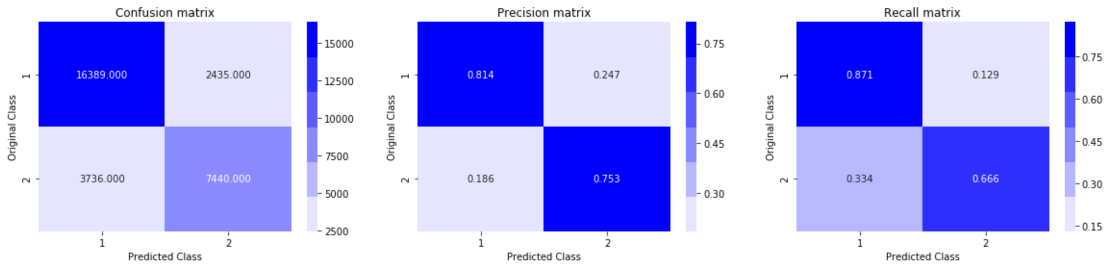
In [34]:

```
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', n_jobs=-1, rando
    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', ranc
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: 0.41571541390792
74
For values of alpha =  0.0001 The log loss is: 0.4385273083526
162
For values of alpha =  0.001 The log loss is: 0.44983101536719
59
For values of alpha =  0.01 The log loss is: 0.523457425488220
4
For values of alpha =  0.1 The log loss is: 0.4921133053859541
For values of alpha =  1 The log loss is: 0.5800516868964605
For values of alpha =  10 The log loss is: 0.6444120726551625
```




```
For values of best alpha =  1e-05 The train log loss is: 0.405
0380423055367
For values of best alpha =  1e-05 The test log loss is: 0.4157
154139079274
Total number of data points : 30000
```



2. Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss.

Preparing Data

In [2]:

```
# avoid decoding problems
df = pd.read_csv("train.csv")

df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))

df.head(2)
```

Out[2]:

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0

In [3]:

```
df.shape
```

Out[3]:

```
(404290, 6)
```

In [4]:

```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile('nlp_features_train.csv'):
    dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
else:
    print("download nlp_features_train.csv from drive or run previous noteboo

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='la
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run p
```

In [5]:

```
df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis
df1.head(2)
```

Out[5]:

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	

In [6]:

```
df2.head(2)
```

Out[6]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Co
0	0	1	1	66	57	14	12	
1	1	4	1	51	88	8	13	

In [7]:

```
df = df.drop(['qid1','qid2'],axis=1)
df.head(2)
```

Out[7]:

	id	question1	question2	is_duplicate
0	0	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0

In [8]:

```
f1  = df1
f2  = f1.merge(df2, on='id',how='left')
X = f2.merge(df, on='id',how='left')
X.head(2)
```

Out[8]:

	id	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_
0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	

2 rows × 30 columns

In [9]:

```
X.shape
```

Out[9]:

```
(404290, 30)
```

In [10]:

```
y_true = X['is_duplicate']
X.drop(['id','is_duplicate'], axis=1, inplace=True)

X.head(2)
```

Out[10]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0

2 rows × 28 columns

In [11]:

```
X.columns
```

Out[11]:

```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min',
'ctc_max',
       'last_word_eq', 'first_word_eq', 'abs_len_diff', 'mean_
len',
       'token_set_ratio', 'token_sort_ratio', 'fuzz_ratio',
       'fuzz_partial_ratio', 'longest_substr_ratio', 'freq_qid
1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Com
mon',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2',
'question1',
       'question2'],
      dtype='object')
```

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(X.head(100000), y_true[0:

print(X_train.shape,len(y_train))
print(X_test.shape, len(y_test))
```

```
(70000, 28) 70000
(30000, 28) 30000
```

Vectorizing text

In [13]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(X_train['question1']) + list(X_train['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(questions)

# dict key:word and value:tf-idf score
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

In [39]:

```
import spacy
from tqdm import tqdm
# tqdm is used to print the progress bar

# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
for qu1 in tqdm(list(X_train['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
```

```
100%|████████████████████████████████████████
██████████| 70000/70000 [07:26<00:00, 156.76it/s]
```

In [41]:

```
X_train_q1 = pd.DataFrame(vecs1)
X_train_q1.head(2)
```

Out[41]:

	0	1	2	3	4	5	
0	174.780943	-1.783925	-95.148558	172.623391	165.402961	331.097888	62.
1	-22.009782	33.274417	10.360699	-23.054704	25.216494	-61.609362	9.

2 rows × 96 columns

In [42]:

```
vecs1 = []
for qu1 in tqdm(list(X_test['question1'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
```

```
100%|██████████████████████████████████████████████████
████████████████| 30000/30000 [03:12<00:00, 156.05it/s]
```

In [43]:

```
X_test_q1 = pd.DataFrame(vecs1)
X_test_q1.head(2)
```

Out[43]:

	0	1	2	3	4	5	
0	43.532679	19.955886	-20.477901	98.345907	43.305930	82.937996	71.4928
1	68.287131	12.319043	2.739216	9.422874	45.560484	27.538594	52.1096

2 rows × 96 columns

In [44]:

```
vecs1 = []
for qu1 in tqdm(list(X_train['question2'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
```

```
100%|██████████████████████████████████████████████████
████████████████| 70000/70000 [07:31<00:00, 155.02it/s]
```

In [45]:

```
X_train_q2 = pd.DataFrame(vecs1)
X_train_q2.head(2)
```

Out[45]:

	0	1	2	3	4	5	
0	23.984274	-92.230291	-236.53875	-79.976596	142.287972	338.977032	214
1	-26.019013	0.522169	-36.26695	-41.558604	13.195390	-31.610790	22

2 rows × 96 columns

In [46]:

```
vecs1 = []
for qu1 in tqdm(list(X_test['question2'])):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
```

```
100%|██████████████████████████████████████████████████
████████████████| 30000/30000 [03:11<00:00, 156.76it/s]
```

In [47]:

```
X_test_q2 = pd.DataFrame(vecs1)
X_test_q2.head(2)
```

Out[47]:

	0	1	2	3	4	5	
0	55.577907	10.484515	-5.015465	28.894426	48.620527	44.875751	66.89
1	58.872852	36.336705	-59.881256	32.008245	78.901105	158.498677	186.32

2 rows × 96 columns

In [48]:

```
X_train = pd.concat([X_train.reset_index(drop=True),X_train_q1.reset_index(dr
X_train.shape
```

Out[48]:

```
(70000, 220)
```

In [49]:

```
X_test = pd.concat([X_test.reset_index(drop=True),X_test_q1.reset_index(drop=
X_test.shape
```

Out[49]:

```
(30000, 220)
```

In [50]:

```
X_train.head(2)
```

Out[50]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.249997	0.142856	0.333328	0.199998	0.285712	0.137931	0.0
1	0.000000	0.000000	0.749981	0.749981	0.599988	0.599988	1.0

2 rows × 220 columns

In [51]:

```
X_test.head(2)
```

Out[51]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.999967	0.599988	0.749981	0.749981	0.857131	0.666659	1.0
1	0.399992	0.166665	0.333328	0.285710	0.363633	0.166666	0.0

2 rows × 220 columns

In [54]:

```
X_train.question1.head(2)
```

Out[54]:

```
0    What is it like to swim nude at Piscine Roger ...
1                                What hurts you the most?
Name: question1, dtype: object
```

In [55]:

```
X_train.drop(['question1','question2'], axis=1, inplace=True)
X_train.head(2)
```

Out[55]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.249997	0.142856	0.333328	0.199998	0.285712	0.137931	0.0
1	0.000000	0.000000	0.749981	0.749981	0.599988	0.599988	1.0

2 rows × 218 columns

In [56]:

```
X_test.drop(['question1','question2'], axis=1, inplace=True)
X_test.head(2)
```

Out[56]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq
0	0.999967	0.599988	0.749981	0.749981	0.857131	0.666659	1.0
1	0.399992	0.166665	0.333328	0.285710	0.363633	0.166666	0.0

2 rows × 218 columns

In [59]:

```
# changing duplicate colum names

cols=pd.Series(X_train.columns)

for dup in cols[cols.duplicated()].unique():
    cols[cols[cols == dup].index.values.tolist()] = [str(dup) + '.' + str(i)

# rename the columns with the cols list.
X_train.columns=cols

X_train
```

Out[59]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_worc
0	0.249997	0.142856	0.333328	0.199998	0.285712	0.137931	
1	0.000000	0.000000	0.749981	0.749981	0.599988	0.599988	
2	0.499975	0.499975	0.999950	0.999950	0.749981	0.749981	
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
4	0.888879	0.799992	0.999991	0.999991	0.904758	0.863632	
...	...	...	...	...	...	...	
69995	0.749981	0.599988	0.499988	0.499988	0.624992	0.555549	
69996	0.499975	0.249994	0.999900	0.999900	0.666644	0.399992	
69997	0.299997	0.249998	0.499992	0.428565	0.352939	0.333331	
69998	0.249994	0.199996	0.000000	0.000000	0.111110	0.090908	
69999	0.249994	0.199996	0.599988	0.428565	0.399996	0.333331	

70000 rows × 218 columns

In [60]:

```
cols=pd.Series(X_test.columns)

for dup in cols[cols.duplicated()].unique():
    cols[cols[cols == dup].index.values.tolist()] = [str(dup) + '.' + str(i)

# rename the columns with the cols list.
X_test.columns=cols

X_test
```

Out[60]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_worc
0	0.999967	0.599988	0.749981	0.749981	0.857131	0.666659	
1	0.399992	0.166665	0.333328	0.285710	0.363633	0.166666	
2	0.999975	0.666656	0.999980	0.999980	0.999989	0.818174	
3	0.999980	0.833319	0.999967	0.749981	0.999988	0.799992	
4	0.799984	0.799984	0.999983	0.999983	0.909083	0.909083	
...	...	...	...	...	...	...	
29995	0.749981	0.749981	0.999950	0.499988	0.833319	0.624992	
29996	0.666644	0.399992	0.000000	0.000000	0.499988	0.285710	
29997	0.249994	0.199996	0.999900	0.199996	0.399992	0.199998	
29998	0.499992	0.499992	0.333322	0.111110	0.444440	0.235293	
29999	0.749981	0.499992	0.999975	0.999975	0.874989	0.699993	

30000 rows × 218 columns

Hyperparameter tuning

In [61]:

```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV
parameters = {'max_depth' : [1, 2, 3, 4, 5, 6], 'eta' : [0.1,0.2,0.3,0.4,0.5,
xgbdt = xgb.XGBClassifier()
clf = GridSearchCV(xgbdt, parameters, cv=5,  return_train_score=True)
clf.fit(X_train[0:1000], y_train[0:1000])

results = pd.DataFrame.from_dict(clf.cv_results_)
results = results.sort_values(['param_max_depth'])

train_auc= results['mean_train_score']
train_auc_std= results['std_train_score']
cv_auc = results['mean_test_score']
cv_auc_std= results['std_test_score']
K =  results['param_max_depth']
M = results['param_eta']
```

In [62]:

```
print(results)
```

```
     mean_fit_time  std_fit_time  mean_score_time  std_scor
e_time  \
0         0.133648      0.037215         0.010970        0.
000001
36        0.104772      0.003723         0.010277        0.
000401
24        0.113298      0.001353         0.010197        0.
000522
102       0.128741      0.010886         0.010572        0.
000798
42        0.115302      0.005287         0.010244        0.
000389
..             ...           ...              ...
...
17        0.330786      0.032365         0.010075        0.
000487
11        0.341377      0.001880         0.010474        0.
000446
5         0.393578      0.007691         0.010391        0.
```

3D plot showing hyper parametrs impact on the model

In [63]:

```
import plotly
import plotly.offline as offline
import plotly.graph_objs as go

trace1 = go.Scatter3d(x = K, y = M, z = train_auc, name = 'Train')
trace2 = go.Scatter3d(x = K, y = M, z = cv_auc, name = 'Cross Validation')
data = [trace1, trace2]

layout = go.Layout(scene = dict(xaxis = dict(title = 'max_depth'), yaxis = di
                               zaxis = dict(title = 'AUC'),))

fig = go.Figure(data = data, layout = layout)
offline.iplot(fig, filename='3d-scatter-colorscale')
```

Best hyper parameters

In [65]:

```
best_max_depth = clf.best_params_['max_depth']
best_eta = clf.best_params_['eta']
best_booster = clf.best_params_['booster']
print('best value for max depth is {} '.format(best_max_depth))
print('best value for eta is {} '.format(best_eta))
print('best value for booster is {} '.format(best_booster))
```

```
best value for max depth is 4
best value for eta is 0.2
best value for booster is gbtree
```

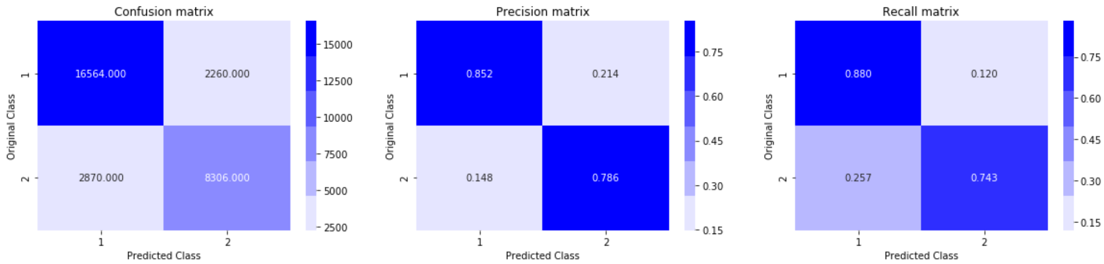
Applying the best hyper parameters to XGBoost model

In [68]:

```
#### import xgboost as xgb
params = {}
params['objective'] = 'binary:logistic'
params['eval_metric'] = 'logloss'
params['booster'] = best_booster
params['eta'] = best_eta
params['max_depth'] = best_max_depth

d_train_new = xgb.DMatrix(X_train, label=y_train)
d_test_new = xgb.DMatrix(X_test, label=y_test)

watchlist = [(d_train_new, 'train'), (d_test_new, 'valid')]

bst = xgb.train(params, d_train_new, 400, watchlist,early_stopping_rounds=20,

xgdmat = xgb.DMatrix(X_train,y_train)
predict_y = bst.predict(d_test_new)
print("The test log loss is:",log_loss(y_test, predict_y, labels=clf.classes_

predicted_y =np.array(predict_y>0.5,dtype=int)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
[0]	train-logloss:0.61759	valid-logloss:0.61814
Multiple eval metrics have been passed: 'valid-logloss' will b
e used for early stopping.

Will train until valid-logloss hasn't improved in 20 rounds.
[10]	train-logloss:0.40338	valid-logloss:0.40738
[20]	train-logloss:0.37267	valid-logloss:0.37950
[30]	train-logloss:0.35732	valid-logloss:0.36668
[40]	train-logloss:0.34806	valid-logloss:0.36036
[50]	train-logloss:0.34025	valid-logloss:0.35521
[60]	train-logloss:0.33402	valid-logloss:0.35295
[70]	train-logloss:0.32837	valid-logloss:0.35068
[80]	train-logloss:0.32254	valid-logloss:0.34853
[90]	train-logloss:0.31777	valid-logloss:0.34714
[100]	train-logloss:0.31410	valid-logloss:0.34637
[110]	train-logloss:0.30963	valid-logloss:0.34530
[120]	train-logloss:0.30528	valid-logloss:0.34475
[130]	train-logloss:0.30168	valid-logloss:0.34407
[140]	train-logloss:0.29736	valid-logloss:0.34329
[150]	train-logloss:0.29402	valid-logloss:0.34336
[160]	train-logloss:0.29019	valid-logloss:0.34309
[170]	train-logloss:0.28744	valid-logloss:0.34303
[180]	train-logloss:0.28384	valid-logloss:0.34288
[190]	train-logloss:0.28080	valid-logloss:0.34257
[200]	train-logloss:0.27780	valid-logloss:0.34279
[210]	train-logloss:0.27472	valid-logloss:0.34251
[220]	train-logloss:0.27213	valid-logloss:0.34232
```

```
[230]	train-logloss:0.26873	valid-logloss:0.34222
[240]	train-logloss:0.26529	valid-logloss:0.34180
[250]	train-logloss:0.26259	valid-logloss:0.34175
[260]	train-logloss:0.26009	valid-logloss:0.34173
[270]	train-logloss:0.25696	valid-logloss:0.34125
[280]	train-logloss:0.25446	valid-logloss:0.34128
[290]	train-logloss:0.25186	valid-logloss:0.34133
Stopping. Best iteration:
[271]	train-logloss:0.25660	valid-logloss:0.34115

The test log loss is: 0.3413284653884588
Total number of data points : 30000
```



Conclusion

In [71]:

```
# http://zetcode.com/python/prettytable/
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ["Vectorizer", "Model", "Hyper Parameters", "Log Loss"]
table.add_row(['TFIDF W2V', 'Logestic Regression', 'Alpha = 0.01', '0.4433'])
table.add_row(['TFIDF W2V', 'Linear SVM', 'Aalpha =  0.01', '0.5896'])
table.add_row(['TFIDF W2V', 'GBDT (XGBoost)', 'eta = 0.02 max_depth = 4 ', '0
table.add_row(['','','',''])
table.add_row(['TFIDF', 'Logestic Regression', 'Alpha = 0.01', '0.4366'])
table.add_row(['TFIDF', 'Linear SVM', 'Alpha =  0.00001', '0.4157'])
table.add_row(['TFIDF W2V', 'GBDT (XGBoost)', 'eta = 0.2 max_depth = 4 booste
print(table)
```

```
+------------+---------------------+------------------------
---------------+----------+
| Vectorizer |        Model        |            Hyper Paramet
ers              | Log Loss |
+------------+---------------------+------------------------
---------------+----------+
| TFIDF W2V  | Logestic Regression |               Alpha = 0.0
1                |  0.4433  |
| TFIDF W2V  |      Linear SVM     |              Aalpha =  0.
01               |  0.5896  |
| TFIDF W2V  |    GBDT (XGBoost)   |        eta = 0.02 max_dep
th = 4           |  0.3546  |
|            |                     |
|          |
|   TFIDF    | Logestic Regression |               Alpha = 0.0
1                |  0.4366  |
|   TFIDF    |      Linear SVM     |             Alpha =  0.00
001              |  0.4157  |
| TFIDF W2V  |    GBDT (XGBoost)   | eta = 0.2 max_depth = 4 b
ooster = gbtree |  0.3411  |
+------------+---------------------+------------------------
---------------+----------+
```

Summary

1. Hyper parameter tuning XGBoost

- Performed Grid search to find the best hyper parameters
- Obtained the best 'eta' as 0.2
- Obtained the best 'max_depth' as 4
- Obtained the best booster as gbtree
- The log-loss decreased from 0.3546 to 0.34115 with obtained best hyper parameters

2. Logestic regression and Linear SVM with TFIDF features

- Replaced TFIDF W2V with TFIDF features, resulting in dimensions change from 794 to 1026
- Performed hyperparameter tuning and found the best alpha for Logestic regression = 0.01 and Linear SVM = 0.00001
- The log-loss decreased from 0.4433 to 0.4366 for Logestic Regression with TFIDF featurization
- The log-loss decreased from 0.4433 to 0.4366 for Linear SVM with TFIDF featurization

Observations

- Performance of models trained with TFIDF features is better than models trained without TFIDF features
- GBDT has the best log-loss