Quora Question Pairs

1. Business Problem

1.1 Description

Quora is a place to gain and share knowledge—about anything. It's a platform to ask questions and connect with people who contribute unique insights and quality answers. This empowers people to learn from each other and to better understand the world.

Over 100 million people visit Quora every month, so it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.

> Credits: Kaggle

Problem Statement

- Identify which questions asked on Quora are duplicates of questions that have already been asked.
- This could be useful to instantly provide answers to questions that have already been answered.
- We are tasked with predicting whether a pair of questions are duplicates or not.

1.2 Sources/Useful Links

- Source : https://www.kaggle.com/c/quora-question-pairs

Useful Links

- Discussions : https://www.kaggle.com/anokas/data-analysis-xgboost-starter-0-35460-lb/comments
- Kaggle Winning Solution and other approaches: https://www.dropbox.com/sh/93968nfnrzh8bp5/AACZdtsApc1QSTQc7X0H3QZ5a?dl=0
- Blog 1 : https://engineering.quora.com/Semantic-Question-Matching-with-Deep-Learning
- Blog 2 : https://towardsdatascience.com/identifying-duplicate-questions-on-quora-top-12-on-kaggle-4c1cf93f1c30

1.3 Real world/Business Objectives and Constraints

1. The cost of a mis-classification can be very high.
2. You would want a probability of a pair of questions to be duplicates so that you can choose any threshold of choice.
3. No strict latency concerns.
4. Interpretability is partially important.

2. Machine Learning Probelm

2.1 Data

2.1.1 Data Overview

- Data will be in a file Train.csv

- Train.csv contains 5 columns : qid1, qid2, question1, question2, is_duplicate
- Size of Train.csv - 60MB
- Number of rows in Train.csv = 404,290

2.1.2 Example Data point

```
"id","qid1","qid2","question1","question2","is_duplicate"
"0","1","2","What is the step by step guide to invest in share market in india?","What is
the step by step guide to invest in share market?","0"
"1","3","4","What is the story of Kohinoor (Koh-i-Noor) Diamond?","What would happen if the
Indian government stole the Kohinoor (Koh-i-Noor) diamond back?","0"
"7","15","16","How can I be a good geologist?","What should I do to be a great
geologist?","1"
"11","23","24","How do I read and find my YouTube comments?","How can I see all my Youtube
comments?","1"
```

2.2 Mapping the real world problem to an ML problem

2.2.1 Type of Machine Leaning Problem

It is a binary classification problem, for a given pair of questions we need to predict if they are duplicate or not.

2.2.2 Performance Metric

Source: https://www.kaggle.com/c/quora-question-pairs#evaluation

Metric(s):

- log-loss : https://www.kaggle.com/wiki/LogarithmicLoss
- Binary Confusion Matrix

2.3 Train and Test Construction

We build train and test by randomly splitting in the ratio of 70:30 or 80:20 whatever we choose as we have sufficient points to work with.

3. Exploratory Data Analysis

In [1]:

```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3 as sqlite
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from chart_studio import plotly
import plotly.offline as offline
import plotly.graph_objs as go
offline.init_notebook_mode()
```

In [2]:

```
dfnlp = pd.read_csv("nlp_features_train.csv",encoding='latin-1')
dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
```

```
df1=dfppro.drop(['qid1', 'qid2', 'question1', 'question2', 'is_duplicate'],axis=1)
data=dfnlp.merge(df1,on="id",how="left")
data['question1'] = data['question1'].apply(lambda x: str(x))
data['question2'] = data['question2'].apply(lambda x: str(x))

data=data.head(50000)
data.fillna('')
print(data.info())
del dfnlp
del dfppro
del df1
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 50000 entries, 0 to 49999
Data columns (total 32 columns):
id                      50000 non-null int64
qid1                    50000 non-null int64
qid2                    50000 non-null int64
question1               50000 non-null object
question2               50000 non-null object
is_duplicate            50000 non-null int64
cwc_min                 50000 non-null float64
cwc_max                 50000 non-null float64
csc_min                 50000 non-null float64
csc_max                 50000 non-null float64
ctc_min                 50000 non-null float64
ctc_max                 50000 non-null float64
last_word_eq            50000 non-null float64
first_word_eq           50000 non-null float64
abs_len_diff            50000 non-null float64
mean_len                50000 non-null float64
token_set_ratio         50000 non-null int64
token_sort_ratio        50000 non-null int64
fuzz_ratio              50000 non-null int64
fuzz_partial_ratio      50000 non-null int64
longest_substr_ratio    50000 non-null float64
freq_qid1               50000 non-null int64
freq_qid2               50000 non-null int64
q1len                   50000 non-null int64
q2len                   50000 non-null int64
q1_n_words              50000 non-null int64
q2_n_words              50000 non-null int64
word_Common             50000 non-null float64
word_Total              50000 non-null float64
word_share              50000 non-null float64
freq_q1+q2              50000 non-null int64
freq_q1-q2              50000 non-null int64
dtypes: float64(14), int64(16), object(2)
memory usage: 12.6+ MB
None
```

Data cleaning

In [3]:

```
from tqdm import tqdm
import re

stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've",
\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his',
'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them',
'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll",
'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having',
'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', '
while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during',
'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
```

```
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under
', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'e
ach', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll'
, 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "d
esn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'mightn',
"mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn',
"wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]


def decontracted(phrase):
    # specific
    phrase = re.sub(r"won't", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)
    return phrase

def textClean(data):
    output=[]
    for line in tqdm(data):
        sent =decontracted(line)
        sent = sent.replace('\\r', ' ')
        sent = sent.replace('\\"', ' ')
        sent = sent.replace('\\n', ' ')
        sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
        sent = re.sub(r"([0-9]+)000000", r"\1m", sent)
        sent = re.sub(r"([0-9]+)000", r"\1k", sent)
        sent = sent.replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar ").repl
ace(",000,000", "m").replace(",000", "k")
        sent=' '.join(e for e in sent.split() if e not in stopwords)
        output.append(sent.lower().strip())
    return output
```

In [4]:

```
data["question1"]=textClean(data["question1"].values)
data["question2"]=textClean(data["question2"].values)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:04<00:00, 10160.30it/s]
100%|██████████████████████████████████████████████████████████████████████████| 50000/50000
[00:05<00:00, 9264.15it/s]
```

In [5]:

```
adv_feat_y=data["is_duplicate"]
data.drop(["id","qid1", "qid2","is_duplicate"],axis=1,inplace=True)
```

Spliting

In [6]:

```
x_train,x_test,y_train,y_test=train_test_split(data, adv_feat_y, test_size=0.3, stratify=adv_feat_y
)
del adv_feat_y
```

Vectorizing

In [9]:

```
import pickle

def tfidf(train,data):
    vec=TfidfVectorizer(ngram_range=(1,2),min_df=15,max_df=5000)
    vec.fit(train,y_train)
    return vec.transform(data)


    #creating dictionary of Tfidf for project_title
tfidf_model_q1= TfidfVectorizer()
tfidf_model_q1.fit(x_train["question1"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_q1 = dict(zip(tfidf_model_q1.get_feature_names(), list(tfidf_model_q1.idf_)))
tfidf_words_q1 = set(tfidf_model_q1.get_feature_names())

#creating dictionary of Tfidf for project_title
tfidf_model_q2 = TfidfVectorizer()
tfidf_model_q2.fit(x_train["question2"])
# we are converting a dictionary with word as a key, and the idf as a value
dictionary_q2 = dict(zip(tfidf_model_q2.get_feature_names(), list(tfidf_model_q2.idf_)))
tfidf_words_q2 = set(tfidf_model_q2.get_feature_names())

#loading pretrained model
# storing variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save
-and-load-variables-in-python/
with open('glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


def tfidfW2v(process_dat, tfidf_mod, dictionary):
    tfidf_w2v_vectors_title = []; # the avg-w2v for each sentence/review is stored in this list
    for sentence in tqdm(process_dat): # for each review/sentence
        vector = np.zeros(300) # as word vectors are of zero length
        tf_idf_weight =0; # num of words with a valid vector in the sentence/review
        for word in sentence.split(): # for each word in a review/sentence
            if (word in glove_words) and (word in tfidf_mod):
                vec = model[word] # getting the vector for each word
                # here we are multiplying idf value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))
                tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting
the tfidf value for each word
                vector += (vec * tf_idf) # calculating tfidf weighted w2v
                tf_idf_weight += tf_idf
        if tf_idf_weight != 0:
            vector /= tf_idf_weight
        tfidf_w2v_vectors_title.append(vector)
    return tfidf_w2v_vectors_title
```

In [10]:

```
q1_tfidf=tfidf(x_train["question1"],x_train["question1"])
q2_tfidf=tfidf(x_train["question2"],x_train["question2"])

print("q1_tfidf train:",q1_tfidf.shape)
print("q2_tfidf train:",q2_tfidf.shape)

q1_tfidf_test=tfidf(x_train["question1"],x_test["question1"])
q2_tfidf_test=tfidf(x_train["question2"],x_test["question2"])

print("q1_tfidf test:",q1_tfidf_test.shape)
print("q2_tfidf test:",q2_tfidf_test.shape)


datatrain=x_train.drop(["question1","question2"],axis=1).reset_index(drop=True)
datatest=x_test.drop(["question1","question2"],axis=1).reset_index(drop=True)
```

```
q1_tfidf train: (35000, 2665)
q2_tfidf train: (35000, 2725)
```

```
q1_tfidf test: (15000, 2665)
q2_tfidf test: (15000, 2725)
```

In [12]:

```
q1_tfidfW2V_train=tfidfW2v(x_train["question1"],tfidf_words_q1,dictionary_q1)
q2_tfidfW2V_train=tfidfW2v(x_train["question2"],tfidf_words_q2,dictionary_q2)

print("q1_tfidfW2V_train train:",len(q1_tfidfW2V_train))
print("q2_tfidfW2V_train train:",len(q2_tfidfW2V_train))

q1_tfidfW2V_test=tfidfW2v(x_test["question1"],tfidf_words_q1,dictionary_q1)
q2_tfidfW2V_test=tfidfW2v(x_test["question2"],tfidf_words_q2,dictionary_q2)

print("q1_tfidfW2V_test test:",len(q1_tfidfW2V_test))
print("q2_tfidfW2V_test test:",len(q2_tfidfW2V_test))
```

```
100%|████████████████████████████████████████████████████████████████| 35000/35000
[00:02<00:00, 13319.02it/s]
100%|████████████████████████████████████████████████████████████████| 35000/35000
[00:02<00:00, 12995.84it/s]

q1_tfidfW2V_train train: 35000
q2_tfidfW2V_train train: 35000

100%|████████████████████████████████████████████████████████████████| 15000/15000
[00:01<00:00, 13767.69it/s]
100%|████████████████████████████████████████████████████████████████| 15000/15000
[00:01<00:00, 13122.32it/s]

q1_tfidfW2V_test test: 15000
q2_tfidfW2V_test test: 15000
```

Stacking

In [15]:

```
train_q1=pd.DataFrame(q1_tfidf.toarray())
train_q2=pd.DataFrame(q2_tfidf.toarray())
train_set = pd.concat([datatrain,train_q1, train_q2], axis = 1)

test_q1=pd.DataFrame(q1_tfidf_test.toarray())
test_q2=pd.DataFrame(q2_tfidf_test.toarray())
test_set = pd.concat([datatest,test_q1, test_q2], axis = 1)

train_w2v_q1=pd.DataFrame(q1_tfidfW2V_train)
train_w2v_q2=pd.DataFrame(q2_tfidfW2V_train)
train_w2v_set = pd.concat([datatrain,train_w2v_q1, train_w2v_q2], axis = 1)

test_w2v_q1=pd.DataFrame(q1_tfidfW2V_test)
test_w2v_q2=pd.DataFrame(q2_tfidfW2V_test)
test_w2v_set = pd.concat([datatest,test_w2v_q1, test_w2v_q2], axis = 1)



del q1_tfidf
del q2_tfidf_test
del q2_tfidf
del q1_tfidf_test
del train_q1
del train_q2
del test_q1
del test_q2
```

Model Defination

In [16]:

```
from sklearn.linear_model import SGDClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics.classification import accuracy_score, log_loss
from sklearn.metrics import confusion_matrix

def logistic_svm(param,x_train,x_test,loss_type,reg):
    log_loss_error=[]
    for i in param:
        model=SGDClassifier(alpha=i,loss=loss_type,penalty=reg ,random_state=42)
        model.fit(x_train,y_train)
        calibration=CalibratedClassifierCV(model,method="sigmoid")
        calibration.fit(x_train,y_train)
        pred_y=calibration.predict_proba(x_test)
        log_loss_error.append(log_loss(y_test, pred_y, labels=model.classes_, eps=1e-15))
        print("alpha: ",i," log loss: ",log_loss(y_test, pred_y, labels=model.classes_, eps=1e-15))

    plt.plot(param,log_loss_error)
    plt.xlabel("Alpha i's")
    plt.ylabel("log loss")
    plt.scatter(param,log_loss_error)
    plt.title("Cross Validation Error for each alpha")
    plt.grid()
    plt.show()

    best_alph=np.argmin(log_loss_error)
    model1=SGDClassifier(alpha=param[best_alph],loss=loss_type,penalty=reg ,random_state=42)
    model1.fit(x_train,y_train)
    calibration1=CalibratedClassifierCV(model1,method="sigmoid")
    calibration1.fit(x_train,y_train)
    pred_y=calibration1.predict_proba(x_train)
    print("best alpha: ",param[best_alph]," log loss: ",log_loss(y_train, pred_y, labels=model1.cla
sses_, eps=1e-15))
    pred_y=calibration1.predict_proba(x_test)
    print("best alpha: ",param[best_alph]," log loss: ",log_loss(y_test, pred_y, labels=model1.clas
ses_, eps=1e-15))
    predicted_y =np.argmax(pred_y,axis=1)
    plotConfusionMat(y_test,predicted_y)

def plotConfusionMat(test_y, predict_y):
    confu_mat = confusion_matrix(test_y, predict_y)
    recall =(((confu_mat.T)/(confu_mat.sum(axis=1))).T)
    precision=(confu_mat/confu_mat.sum(axis=0))
    plt.figure(figsize=(20,4))

    labels = [0,1]
    # representing A in heatmap format
    cmap=sns.light_palette("green")
    plt.subplot(1, 3, 1)
    sns.heatmap(confu_mat, 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(precision, 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(recall, annot=True, cmap=cmap, fmt=".3f", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

Logistic with TFIDF

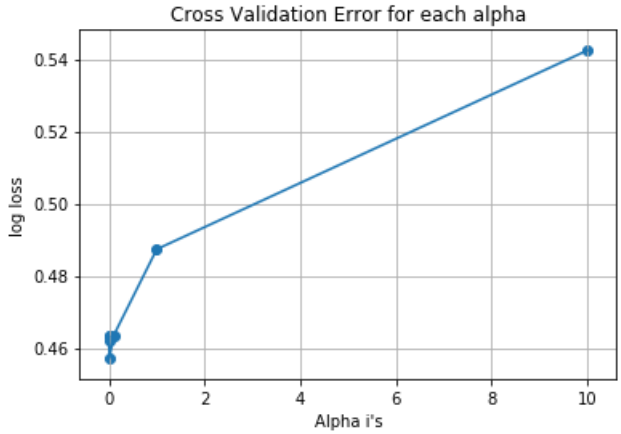
In [11]:

```
alpha = [10 ** x for x in range(-5, 2)]
```

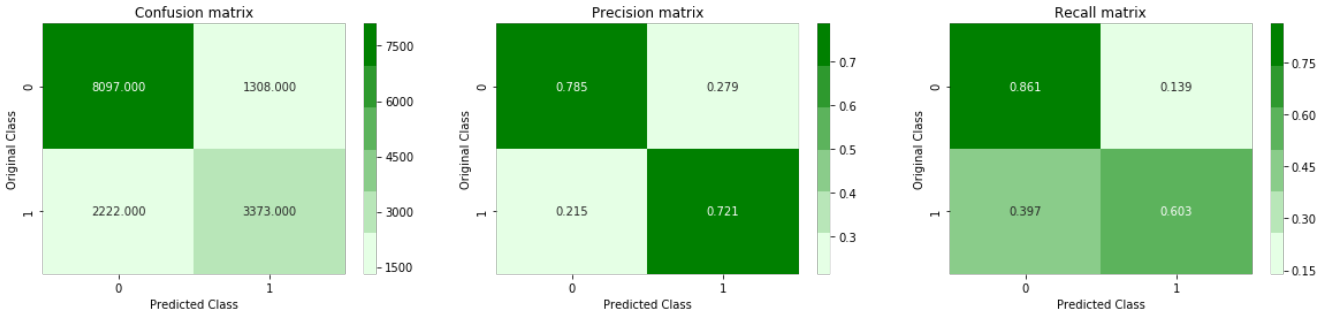
```
alpha = [10 ** x for x in range(-5, 2)]

logistic_svm(alpha,train_set,test_set,"log","l2")
```

```
alpha:  1e-05  log loss:  0.4633209236921876
alpha:  0.0001  log loss:  0.4633230247065496
alpha:  0.001  log loss:  0.4618666257555406
alpha:  0.01  log loss:  0.4571240632617093
alpha:  0.1  log loss:  0.4634734244868585
alpha:  1  log loss:  0.4873276880062743
alpha:  10  log loss:  0.5424265102521505
```




```
best alpha:  0.01  log loss:  0.45606985272335404
best alpha:  0.01  log loss:  0.4571240632617093
```

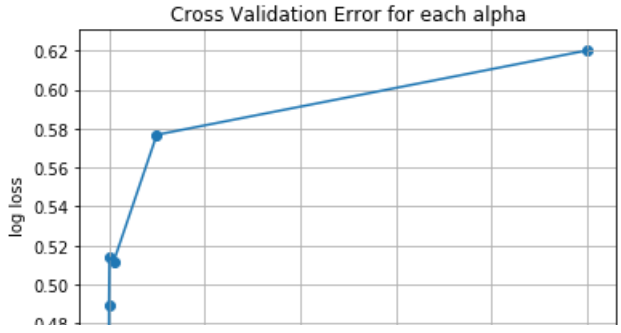


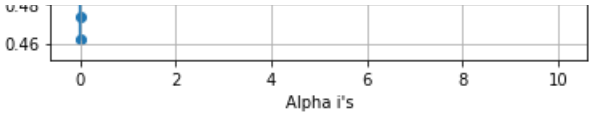
SVM with TFIDF

In [12]:

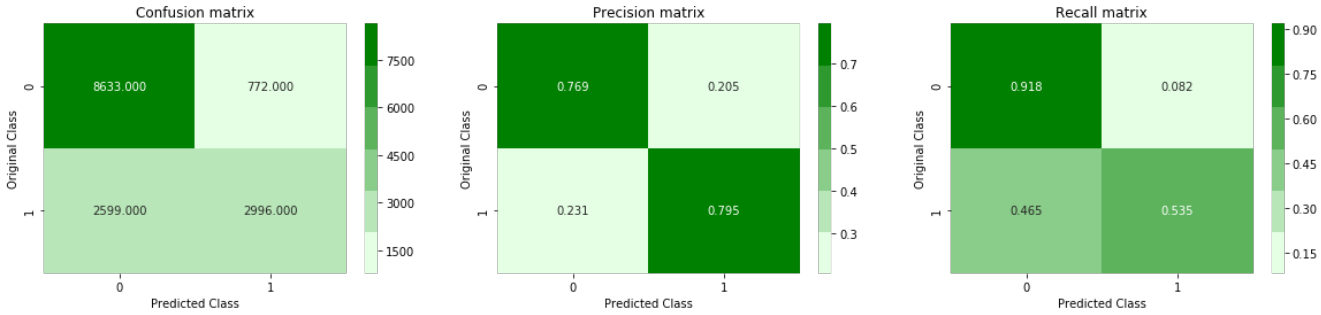
```
logistic_svm(alpha,train_set,test_set,"hinge","l1")
```

```
alpha:  1e-05  log loss:  0.46210688335132216
alpha:  0.0001  log loss:  0.4733176236631921
alpha:  0.001  log loss:  0.4896990271033524
alpha:  0.01  log loss:  0.5138555289045431
alpha:  0.1  log loss:  0.5118347700627717
alpha:  1  log loss:  0.5768467802053875
alpha:  10  log loss:  0.6201709263960006
```






```
best alpha:  1e-05  log loss:  0.46098943976411527
best alpha:  1e-05  log loss:  0.46210688335132216
```



XGBoost with hyperparameter

In [23]:

```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV

def enable_plotly_in_cell():
  import IPython
  from plotly.offline import init_notebook_mode
  display(IPython.core.display.HTML('''<script src="/static/components/requirejs/require.js"></scr
ipt>'''))
  init_notebook_mode(connected=False)

def xgboost(param,x_train,x_test):
    model=xgb.XGBClassifier(objective='binary:logistic')
    grid=RandomizedSearchCV(model,param,n_iter=10,cv=2,scoring='neg_log_loss',n_jobs =-1, return_tr
ain_score=True,refit=True)
    grid.fit(x_train,y_train[0:20000])
    reslt = grid.cv_results_
    print(grid.best_estimator_ )
    print(grid.best_params_ )
    dt_frm = pd.DataFrame.from_dict(reslt)
    trace1 = go.Scatter3d(x=reslt['param_n_estimators'],y=reslt['param_max_depth'],z=reslt['mean_tr
ain_score'], name = 'train')
    trace2 = go.Scatter3d(x=reslt['param_n_estimators'],y=reslt['param_max_depth'],z=reslt['mean_te
st_score'], name = 'Cross validation')
    data = [trace1, trace2]
    enable_plotly_in_cell()

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

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

    pred_y=grid.predict_proba(x_test)

    predicted_y =np.argmax(pred_y,axis=1)
    plotConfusionMat(y_test[0:10000],predicted_y)
```

In [19]:

```
train_w2v_set.shape
```

Out[19]:

```
(35000, 626)
```

In [20]:

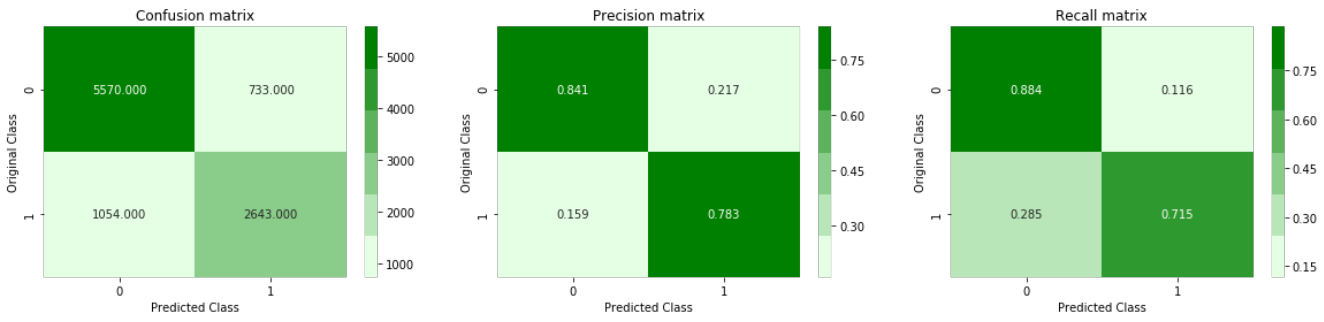
```
li=[i for i in range(0,626)]
train_w2v_set.columns=li
test_w2v_set.columns=li
```

In [24]:

```
#Due to memory constraints dataset has been reduced to 30k points
param = {'n_estimators' :[ 200, 300, 500, 1000], 'max_depth': [ 3, 4, 5, 6, 7]}

xgboost(param,train_w2v_set.head(20000),test_w2v_set.head(10000))
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bynode=1, colsample_bytree=1, gamma=0, learning_rate=0.1,
       max_delta_step=0, max_depth=4, min_child_weight=1, missing=None,
       n_estimators=200, n_jobs=1, nthread=None,
       objective='binary:logistic', random_state=0, reg_alpha=0,
       reg_lambda=1, scale_pos_weight=1, seed=None, silent=None,
       subsample=1, verbosity=1)
{'n_estimators': 200, 'max_depth': 4}
```



In [ ]:

```
```