In [0]:

```
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
```

4. Machine Learning Models

4.1 Reading data from file

In [0]:

```
X_train=pd.read_csv('X_train.csv')
X_test=pd.read_csv('X_test.csv')
```

In [0]:

```
y_train=np.load('y_train.npy')
y_test=np.load('y_test.npy')
```

In [0]:

```
print(type(y_train[0]))
```

```
<class 'numpy.int64'>
```

In [0]:

```
cols = list(X_train.columns)
for i in cols:
    #data[i] = data[i].apply(pd.to_numeric(errors='coerce'))
    X_train[i] = pd.to_numeric(X_train[i],errors='coerce')
cols = list(X_test.columns)
for i in cols:
    #data[i] = data[i].apply(pd.to_numeric(errors='coerce'))
    X_test[i] = pd.to_numeric(X_test[i],errors='coerce')
```

In [6]:

```
X_train=X_train.drop(["q1_feats_m","q2_feats_m"],axis='columns')
X_train.head()
```

Out[6]:

	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
0	0.999950	0.666644	0.499988	0.499988	0.666656	0.571420	0.0	1.0	1.0	6.5	85
1	0.999975	0.999975	0.999980	0.714276	0.999989	0.818174	1.0	1.0	2.0	10.0	100
2	0.874989	0.874989	0.999986	0.999986	0.736838	0.736838	1.0	1.0	0.0	19.0	96
3	0.000000	0.000000	0.499988	0.133332	0.181817	0.068965	0.0	0.0	18.0	20.0	31
4	0.999967	0.374995	0.999967	0.374995	0.999983	0.374998	0.0	1.0	10.0	11.0	100

5 rows × 218 columns

In [0]:

```
X_test=X_test.drop(["q1_feats_m","q2_feats_m"],axis='columns')
```

In [8]:

```
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 : (283003, 218)
Number of data points in test data : (121287, 218)
```

In [0]:

```
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 test data", "-"*10)
test_distr = Counter(y_test)

test_len = len(y_test)
print("Class 0: ",int(test_distr[0])/test_len, "Class 1: ",int(test_distr[1])/test_len)
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6308025003268517 Class 1:  0.36919749967314835
---------- Distribution of output variable in test data ----------
Class 0:  0.6308013224830361 Class 1:  0.3691986775169639
```

In [0]:

```
# This function plots the confusion matrices given y_i, y_i_hat.
```

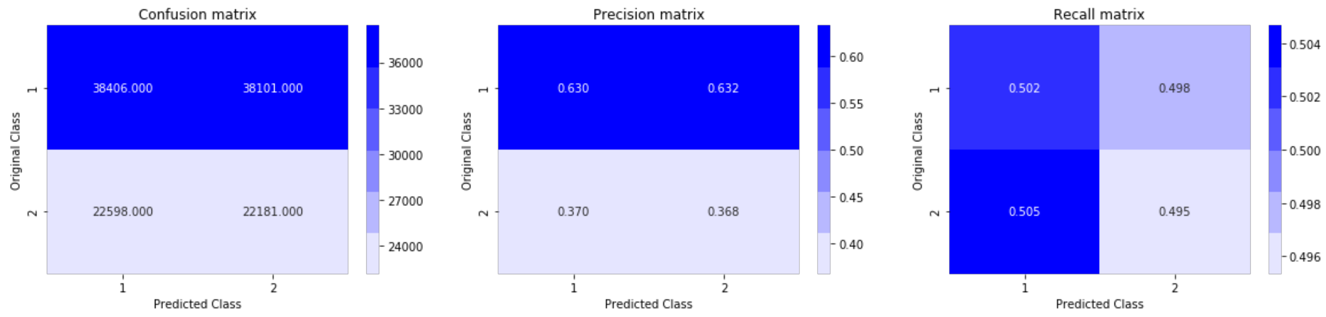
```python
# This function plots the confusion matrices given y_i, y_i_hat.
def plot_confusion_matrix(test_y, predict_y):
    C = confusion_matrix(test_y, predict_y)
    # C = 9,9 matrix, each cell (i,j) represents number of points of class i are predicted class j
    
    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion 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 
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
    
    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 
diamensional array
    # 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=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.show()
```

4.4 Building a random model (Finding worst-case log-loss)

In [0]:

```python
# 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
# we create a output array that has exactly same size as the CV data
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, eps=1e-15))

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

```
Log loss on Test Data using Random Model 0.8876560138945759
```



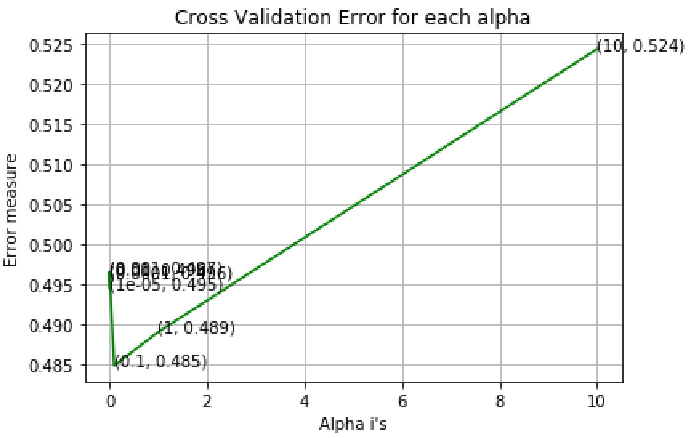
4.4 Logistic Regression with hyperparameter tuning

In [0]:

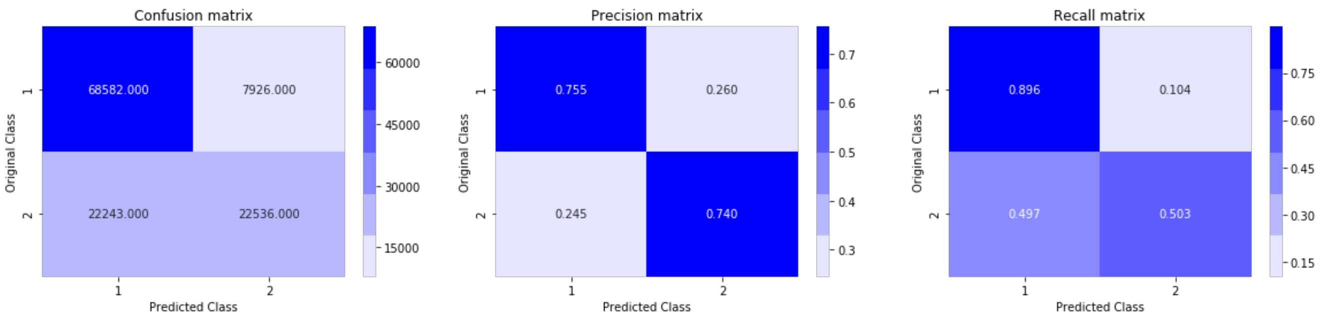
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# 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 Descent.
# 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, labels=clf.cl
asses_, eps=1e-15))

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_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(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
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_test, p
redict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
```

```
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: 0.4945826544651165
For values of alpha =  0.0001 The log loss is: 0.4958703554524966
For values of alpha =  0.001 The log loss is: 0.4965531911676105
For values of alpha =  0.01 The log loss is: 0.4962648672769468
For values of alpha =  0.1 The log loss is: 0.48490252796273603
For values of alpha =  1 The log loss is: 0.48907784493803375
For values of alpha =  10 The log loss is: 0.5243049255886785
```




```
For values of best alpha =  0.1 The train log loss is: 0.48341985453909103
For values of best alpha =  0.1 The test log loss is: 0.48490252796273603
Total number of data points : 121287
```



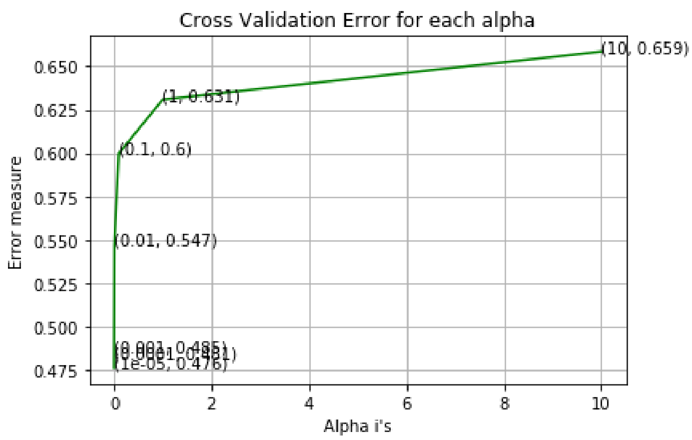
4.5 Linear SVM with hyperparameter tuning

In [0]:

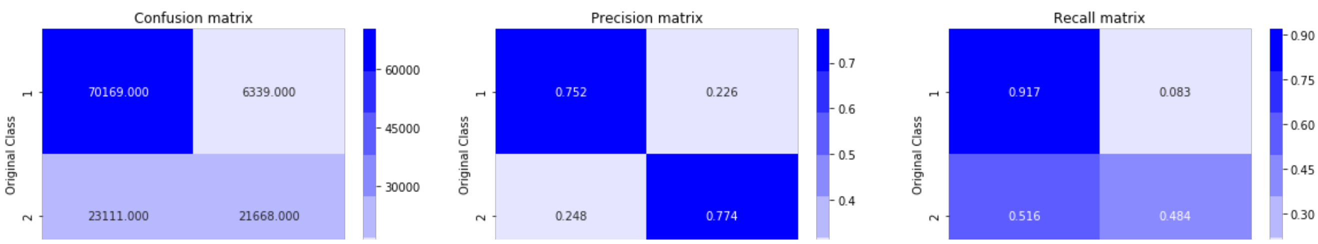
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# 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 Descent.
# predict(X) Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
```

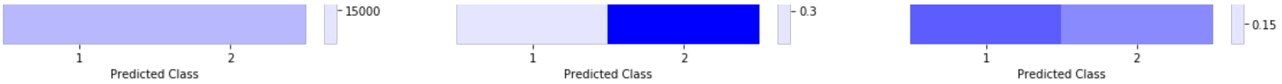
```
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_train, y_train)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_train, y_train)
    predict_y = sig_clf.predict_proba(X_test)
    log_error_array.append(log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, predict_y, labels=clf.cl
asses_, eps=1e-15))

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', 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(y_train,
predict_y, labels=clf.classes_, eps=1e-15))
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_test, p
redict_y, labels=clf.classes_, eps=1e-15))
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: 0.47600341315624184
For values of alpha =  0.0001 The log loss is: 0.4814097437228697
For values of alpha =  0.001 The log loss is: 0.48516914373199327
For values of alpha =  0.01 The log loss is: 0.5470116859115637
For values of alpha =  0.1 The log loss is: 0.5996904083859868
For values of alpha =  1 The log loss is: 0.6309020897451882
For values of alpha =  10 The log loss is: 0.6585278256347589
```




```
For values of best alpha =  1e-05 The train log loss is: 0.4760356569721059
For values of best alpha =  1e-05 The test log loss is: 0.47600341315624184
Total number of data points : 121287
```





4.6 XGBoost

In [0]:

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

param={
'learning_rate':[0.001,0.05,0.5,1],
'n_estimators':[25,50,100,200,500]
}
x_model = xgb.XGBClassifier(n_jobs=-1)
clf = RandomizedSearchCV(x_model,param, cv=2,n_iter=10)
clf.fit(X_train,y_train)
```

In [16]:

```
print(clf.best_params_)
```

```
{'n_estimators': 500, 'learning_rate': 0.5}
```

In [18]:

```
x_model=xgb.XGBClassifier(n_estimators=500,learning_rate=0.5).fit(X_train,y_train)


#y_pred = [int(value) for value in predict_y]
#print(y_pred)
'''
params={
'min_child_weight':5,
'max_depth':10,
'learning_rate':0.05,
'subsample':0.8,
'n_estimators':100
}


d_train = xgb.DMatrix(X_train, label=y_train)
d_test = xgb.DMatrix(X_test, label=y_test)

watchlist = [(d_train, 'train'), (d_test, 'valid')]

bst = xgb.train(params, d_train, 60, watchlist, early_stopping_rounds=20, verbose_eval=10)

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

Out[18]:

```
"\nparams=
{\n'min_child_weight':5,\n'max_depth':10,\n'learning_rate':0.05,\n'subsample':0.8,\n'n_estimators':
n}\n\n\nd_train = xgb.DMatrix(X_train, label=y_train)\nd_test = xgb.DMatrix(X_test,
label=y_test)\n\nwatchlist = [(d_train, 'train'), (d_test, 'valid')]\n\nbst = xgb.train(params, d_
train, 60, watchlist, early_stopping_rounds=20, verbose_eval=10)\n\nxgdmat =
xgb.DMatrix(X_train,y_train)\npredict_y = bst.predict(d_test)\n"
```

In [19]:

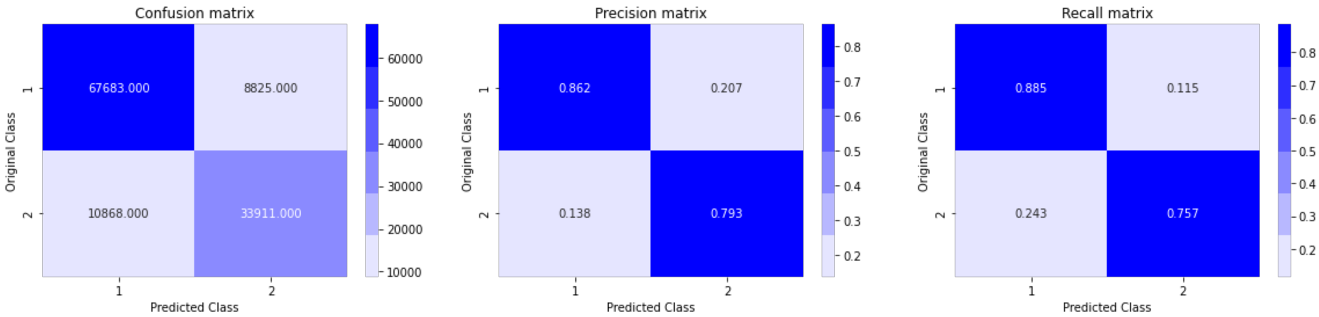
```
predict_y = x_model.predict_proba(X_test)
print("The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
```

```
The test log loss is: 0.3299497664142572
```

In [20]:

```
#predicted_y =np.array(predict_y>0.5,dtype=int)
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```

```
Total number of data points : 121287
```



5. Assignments

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Perform hyperparameter tuning of XgBoost models using RandomsearchCV with vectorizer as TF-IDF W2V to reduce the log-loss.

In [0]:

```
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 notebook")

if os.path.isfile('df_fe_without_preprocessing_train.csv'):
    dfppro = pd.read_csv("df_fe_without_preprocessing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

In [0]:

```
df3 = dfnlp[['id','question1','question2']]
duplicate=dfnlp['is_duplicate']
df1 = dfnlp.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [0]:

```
df1.columns
```

Out[0]:

```
Index(['id', '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'],
      dtype='object')
```

In [0]:

```
df2.columns
```

Out[0]:

```
Index(['id', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share', 'freq_q1+q2',
       'freq_q1-q2'],
```

```
      dtype='object')
```

In [0]:

```
df3.columns
```

Out[0]:

```
Index(['id', 'question1', 'question2'], dtype='object')
```

In [0]:

```
df4=pd.DataFrame()
df3 = df3.fillna(' ')
df4['Text']=df3['question1']+' '+df3['question2']
df4.columns
```

Out[0]:

```
Index(['Text'], dtype='object')
```

In [0]:

```
df4['id']=df1['id']
df1  = df1.merge(df2, on='id',how='left')
result  = df1.merge(df4, on='id',how='left')
```

In [0]:

```
result.columns
```

Out[0]:

```
Index(['id', '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_qid1', 'freq_qid2',
       'q1len', 'q2len', 'q1_n_words', 'q2_n_words', 'word_Common',
       'word_Total', 'word_share', 'freq_q1+q2', 'freq_q1-q2', 'Text'],
      dtype='object')
```

In [0]:

```
result = result.drop('id',axis=1)
```

In [0]:

```
result.shape
```

Out[0]:

```
(404290, 27)
```

In [0]:

```
X_train1,X_test1, y_train1, y_test1 = train_test_split(result, duplicate, stratify=duplicate, test_
size=0.3)
```

In [0]:

```
tfidf = TfidfVectorizer(ngram_range=(1,3),max_features=200000,min_df=0.000032)
text_train = tfidf.fit_transform(X_train1['Text'])
text_test = tfidf.transform(X_test1['Text'])
```

In [0]:

```
print('No of Tfidf features',len(tfidf.get_feature_names()))
```

```
No of Tfidf features 122829
```

In [0]:

```
X_train1=X_train1.drop('Text',axis=1)
X_test1=X_test1.drop('Text',axis=1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-208-ea61df120b5c> in <module>
      4 X_test1=X_test1.drop('Text',axis=1)
      5 X_trainnew=hstack((X_train1,text_train),format='csr',dtype='float64')
----> 6 X_testnew=hstack(X_test1,text_test)

~\Anaconda3\lib\site-packages\scipy\sparse\construct.py in hstack(blocks, format, dtype)
    462
    463     """
--> 464     return bmat([blocks], format=format, dtype=dtype)
    465
    466

~\Anaconda3\lib\site-packages\scipy\sparse\construct.py in bmat(blocks, format, dtype)
    542     """
    543
--> 544     blocks = np.asarray(blocks, dtype='object')
    545
    546     if blocks.ndim != 2:

~\Anaconda3\lib\site-packages\numpy\core\numeric.py in asarray(a, dtype, order)
    499
    500     """
--> 501     return array(a, dtype, copy=False, order=order)
    502
    503

ValueError: cannot copy sequence with size 121287 to array axis with dimension 26
```

In [0]:

```
X_trainnew=hstack((X_train1,text_train),format='csr',dtype='float64')
X_testnew=hstack((X_test1,text_test),format='csr',dtype='float64')
```

In [0]:

```
print(X_trainnew.shape)
print(X_testnew.shape)
```

```
(283003, 122938)
(121287, 122938)
```

In [0]:

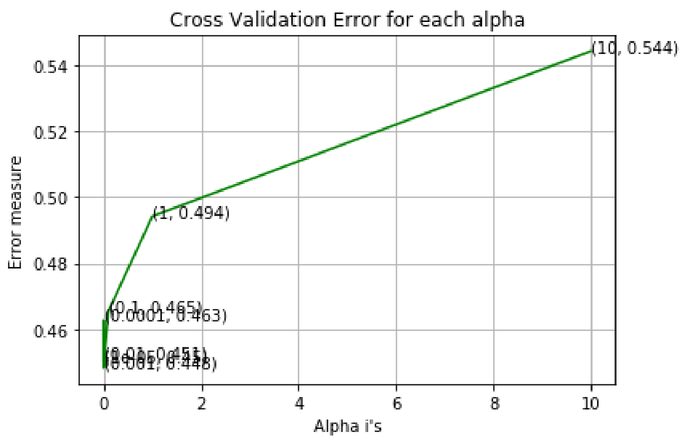
```

```

In [0]:

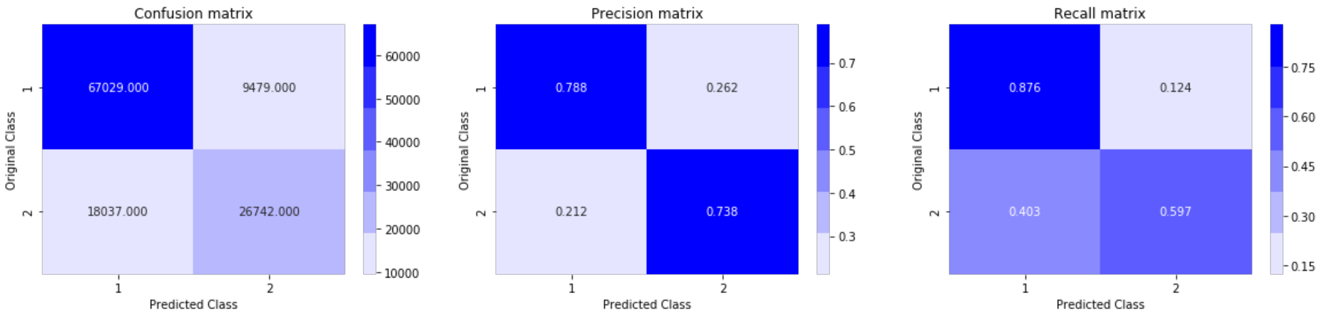
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifier.

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# 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 Descent.
# 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_trainnew, y_train1)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_trainnew, y_train1)
    predict_y = sig_clf.predict_proba(X_testnew)
    log_error_array.append(log_loss(y_test1, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test1, predict_y, labels=clf.c
lasses_, eps=1e-15))

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_state=42)
clf.fit(X_trainnew, y_train1)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_trainnew, y_train1)

predict_y = sig_clf.predict_proba(X_trainnew)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train1
, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_testnew)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test1,
predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test1, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.450424385983166625
For values of alpha =  0.0001 The log loss is: 0.46270711625089794
For values of alpha =  0.001 The log loss is: 0.4482438390679735
For values of alpha =  0.01 The log loss is: 0.4510734270651453
For values of alpha =  0.1 The log loss is: 0.4654342887126676
For values of alpha =  1 The log loss is: 0.49422086781388497
For values of alpha =  10 The log loss is: 0.5442731775912586
```



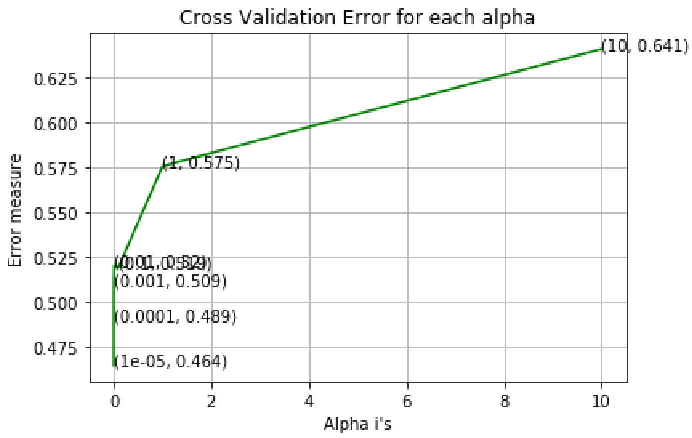

```
For values of best alpha =  0.001 The train log loss is: 0.4480849598389523
For values of best alpha =  0.001 The test log loss is: 0.4482438390679735
Total number of data points : 121287
```



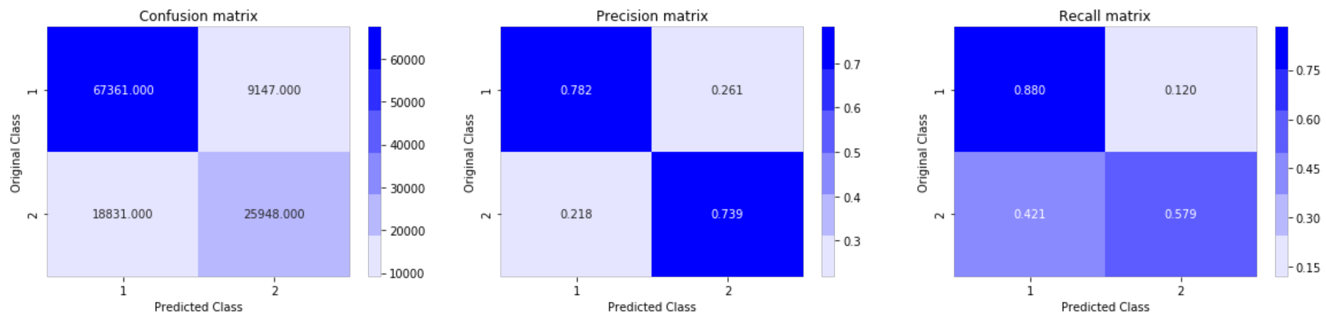
In [0]:

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

# read more about SGDClassifier() at http://scikit-
learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_i
ter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, learning_rate='optimal', eta0
=0.0, power_t=0.5,
# 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 Descent.
# predict(X) Predict class labels for samples in X.

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


log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l1', loss='hinge', random_state=42)
    clf.fit(X_trainnew, y_train1)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(X_trainnew, y_train1)
    predict_y = sig_clf.predict_proba(X_testnew)
    log_error_array.append(log_loss(y_test1, predict_y, labels=clf.classes_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test1, predict_y, labels=clf.c
lasses_, eps=1e-15))

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', random_state=42)
clf.fit(X_trainnew, y_train1)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(X_trainnew, y_train1)

predict_y = sig_clf.predict_proba(X_trainnew)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train1
, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(X_testnew)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:",log_loss(y_test1,
predict_y, labels=clf.classes_, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test1, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.46410137191984074
For values of alpha =  0.0001 The log loss is: 0.4892468548202167
For values of alpha =  0.001 The log loss is: 0.5088646402242227
For values of alpha =  0.01 The log loss is: 0.519977698240258
For values of alpha =  0.1 The log loss is: 0.5188775591389386
For values of alpha =  1 The log loss is: 0.575352958141306
For values of alpha =  10 The log loss is: 0.640598452884727
```




```
For values of best alpha =  1e-05 The train log loss is: 0.46298667602844384
For values of best alpha =  1e-05 The test log loss is: 0.46410137191984074
Total number of data points : 121287
```



In [0]: