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.cross_validation 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
from sklearn.model_selection import train_test_split
from scipy.sparse import hstack
from sklearn.preprocessing import StandardScaler
```

In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

In [0]:
```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
if os.path.isfile("drive/My Drive/Quora/nlp_features_train.csv"):
    dfnlp = pd.read_csv("drive/My Drive/Quora/nlp_features_train.csv",e
ncoding='latin-1')
else:
    print("download nlp_features_train.csv from drive or run previous n
otebook")

if os.path.isfile("drive/My Drive/Quora/df_fe_without_preprocessing_tra
in.csv"):
    dfppro = pd.read_csv("drive/My Drive/Quora/df_fe_without_preprocess
ing_train.csv",encoding='latin-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or
 run previous notebook")
```

In [0]:
```
dfnlp.columns
```

Out[0]:
```
Index(['id', 'qid1', 'qid2', 'question1', 'question2', 'is_duplicate',
       'cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_ma
x',
       '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]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)#it has
 all advanced features
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'
],axis=1)#it has all basic features
#df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],a
xis=1)
#df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.inde
x)
#df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.inde
x)
df3=dfnlp[['id','question1','question2']]
df3['question1'] = df3['question1'].apply(lambda x: str(x))
df3['question2'] = df3['question2'].apply(lambda x: str(x))
```

In [0]:
```
print(df1.columns)
print(df2.columns)
print(df3.columns)
```

```
Index(['id', 'is_duplicate', 'cwc_min', 'cwc_max', 'csc_min', 'csc_ma
x',
       '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')
Index(['id', 'freq_qid1', 'freq_qid2', 'q1len', 'q2len', 'q1_n_words',
       'q2_n_words', 'word_Common', 'word_Total', 'word_share', 'freq_q
1+q2',
       'freq_q1-q2'],
      dtype='object')
Index(['id', 'question1', 'question2'], dtype='object')
```

In [0]:
```
df4=pd.DataFrame()
df4['id']=df3['id']
df4['text']=df3['question1']+' '+df3['question2']
```

In [0]:
```
df1= df1.merge(df2, on='id',how='left')
X= df1.merge(df4, on='id',how='left')
Y=X.is_duplicate
X=X.drop(['is_duplicate','id'],axis=1)
```

In [0]:
```
X.columns
```

Out[0]:
```
Index(['cwc_min', 'cwc_max', 'csc_min', 'csc_max', 'ctc_min', 'ctc_ma
x',
       '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]:
```
#taking only top 100k datapoints
X=X[0:100000]
Y=Y[0:100000]
```

In [0]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3)
tf_idf_vect = TfidfVectorizer()
tf_idf_vect.fit(X_train.text)
X_train_tf_idf = tf_idf_vect.transform(X_train.text)
X_test_tf_idf = tf_idf_vect.transform(X_test.text)
```

In [0]:
```
X_train=X_train.drop(['text'],axis=1)
X_test=X_test.drop(['text'],axis=1)
```

In [0]:
```
a=np.array(X_train_tf_idf)
```

In [0]:
```
print(type(X_train.values))
print(type(X_train_tf_idf))
```

```
<class 'numpy.ndarray'>
<class 'scipy.sparse.csr.csr_matrix'>
```

In [0]:
```
X_train= hstack((X_train.values,X_train_tf_idf))
X_test= hstack((X_test.values,X_test_tf_idf))
```

In [0]:
```
#below is the function to vectorize questions and returns the stacked x
train ,xtest and xcv
def vectorize_and_add_to_df(X,Y):
  X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=
0.33) # this is random splitting

  #applying tfidf vectorizer
  tf_idf_vect = TfidfVectorizer()
  tf_idf_vect.fit(X_train.text)
  X_train_tf_idf = tf_idf_vect.transform(X_train.text)
  X_test_tf_idf = tf_idf_vect.transform(X_test.text)

  #removing Test columns from original dataframe
  X_train=X_train.drop(['text'],axis=1)
  X_test=X_test.drop(['text'],axis=1)

  #stacking tfidf features with old features
  X_train= hstack((X_train.values,X_train_tf_idf))
  X_test= hstack((X_test.values,X_test_tf_idf))

  scale = StandardScaler(with_mean=False)
  X_train = scale.fit_transform(X_train)
  X_test = scale.transform(X_test)
  return X_train,X_test,y_train,y_test
```

In [0]:
```
X_train,X_test,y_train,y_test=vectorize_and_add_to_df(X,Y)
```

```
In [0]: print(X_train.shape)
        print(X_test.shape)
```

```
(67000, 38138)
(33000, 38138)
```

Task 1

confusion matrix

```
In [0]: # 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 cl
        ass i are predicted class j

            A =(((C.T)/(C.sum(axis=1))).T)
            #divid each element of the confusion matrix with the sum of element
        s 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 correspo
        nds 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 element
```

```
s in that row
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 correspo
nds 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()
```

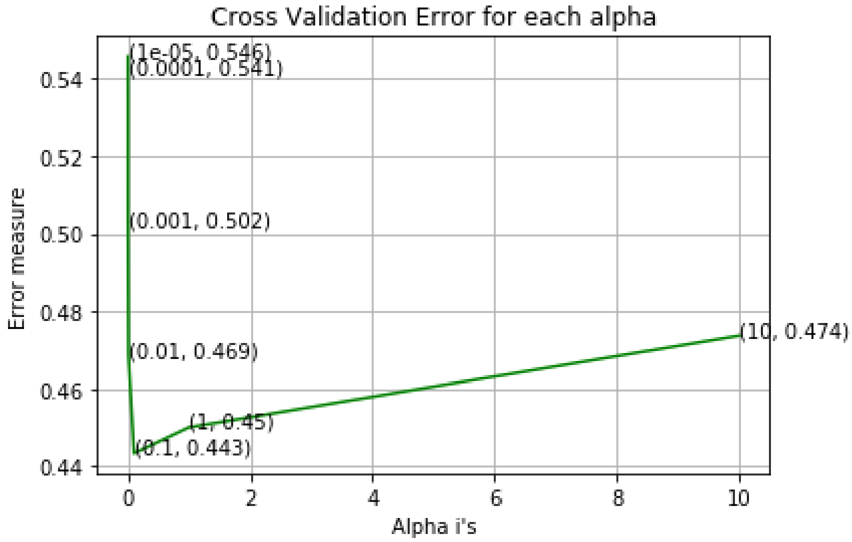
applying logistic regrssion

In [0]:

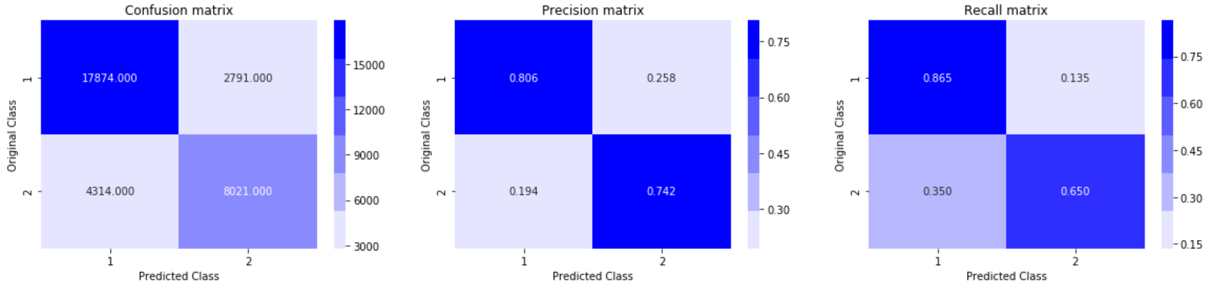
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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,class_weight='balanced')
    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.class
es_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
st, predict_y, labels=clf.classes_, 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,class_weight='balanced')
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 l
oss is:",log_loss(y_test, 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_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.5457465276258013
For values of alpha =  0.0001 The log loss is: 0.5414120671282686
For values of alpha =  0.001 The log loss is: 0.5019773452222286
For values of alpha =  0.01 The log loss is: 0.4685356713336321
For values of alpha =  0.1 The log loss is: 0.4433669178735429
For values of alpha =  1 The log loss is: 0.4500743119579782
For values of alpha =  10 The log loss is: 0.47369403976236374
```




```
For values of best alpha =  0.1 The train log loss is: 0.30358829471896
775
For values of best alpha =  0.1 The test log loss is: 0.443366917873542
9
Total number of data points : 33000
```



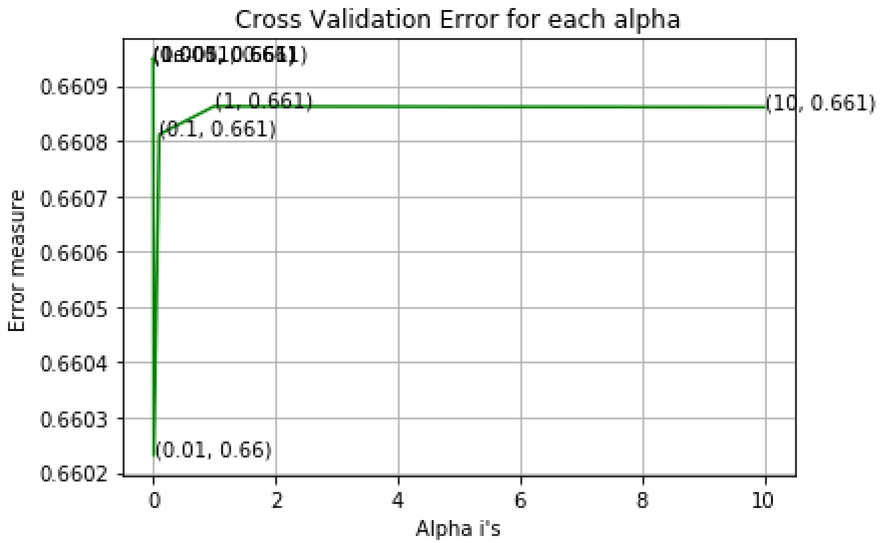
Applying linear svm

In [0]:

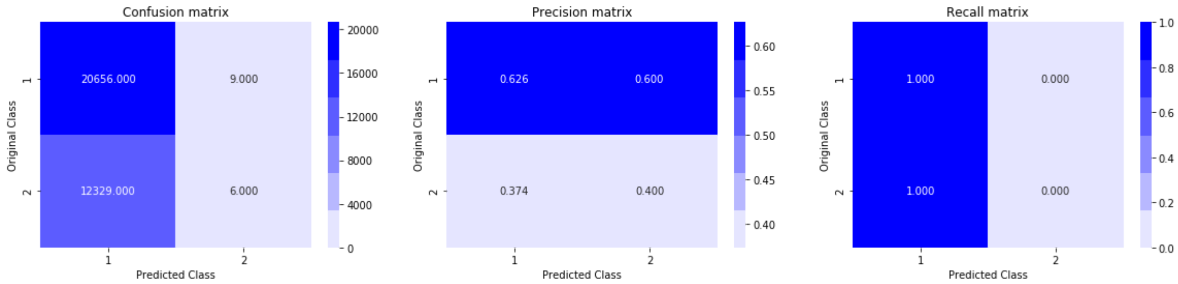
```
alpha = [10 ** x for x in range(-5, 2)] # hyperparam for SGD classifie
r.

# read more about SGDClassifier() at http://scikit-learn.org/stable/mod
ules/generated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.1
5, fit_intercept=True, max_iter=None, tol=None,
# shuffle=True, verbose=0, epsilon=0.1, n_jobs=1, random_state=None, le
arning_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 S
tochastic 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_sta
te=42,class_weight='balanced')
    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.class
es_, eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_te
st, predict_y, labels=clf.classes_, 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,class_weight='balanced')
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 l
oss is:",log_loss(y_test, 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_test, predicted_y)
```

```
For values of alpha =  1e-05 The log loss is: 0.6609482486976545
For values of alpha =  0.0001 The log loss is: 0.6609482486976545
For values of alpha =  0.001 The log loss is: 0.6609482486976545
For values of alpha =  0.01 The log loss is: 0.6602315214861881
For values of alpha =  0.1 The log loss is: 0.6608115679736447
For values of alpha =  1 The log loss is: 0.660862418465362
For values of alpha =  10 The log loss is: 0.6608607643388914
```




```
For values of best alpha =  0.01 The train log loss is: 0.6589370289953
119
For values of best alpha =  0.01 The test log loss is: 0.66023152148618
81
Total number of data points : 33000
```



Task 2

In [0]:
```
#X and Y are 100k points that i am considering
print(X.shape)
print(Y.shape)
```

```
(100000, 27)
(100000,)
```

In [0]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3
3) # this is random splitting
```

In [0]:
```
#this is for train data
i=0
list_of_sentance_train=[]
for sentance in X_train.text:
    list_of_sentance_train.append(sentance.split())

from tqdm import tqdm

# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
model = TfidfVectorizer()
tf_idf_matrix = model.fit_transform(X_train.text)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

from gensim.models import Word2Vec
from gensim.models import KeyedVectors
w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50, workers=
4)
```

```python
w2v_words = list(w2v_model.wv.vocab)

sent_vectors_train = []; # the tfidf-w2v for each sentence/review is st
ored in this list
row=0;
for sent in tqdm(list_of_sentance_train): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    sent_vectors_train.append(sent_vec)
    row += 1
```

```
100%|██████████| 67000/67000 [11:54<00:00, 100.17it/s]
```

```
(67000, 50)
```

In [0]:

```python
tfidf_sent_vectors_train= np.array(sent_vectors_train)
print(tfidf_sent_vectors_train.shape)
```

```
(67000, 50)
```

In [0]:

```python
#this is for test data
i=0
list_of_sentance_test=[]
for sentance in X_test.text:
    list_of_sentance_test.append(sentance.split())
```

```
# S = ["abc def pqr", "def def def abc", "pqr pqr def"]
#model = TfidfVectorizer()
tf_idf_matrix = model.transform(X_test.text)
# we are converting a dictionary with word as a key, and the idf as a v
alue
dictionary = dict(zip(model.get_feature_names(), list(model.idf_)))



# TF-IDF weighted Word2Vec
tfidf_feat = model.get_feature_names() # tfidf words/col-names
# final_tf_idf is the sparse matrix with row= sentence, col=word and ce
ll_val = tfidf

sent_vectors_test = []; # the tfidf-w2v for each sentence/review is sto
red in this list
row=0;
for sent in tqdm(list_of_sentance_test): # for each review/sentence
    sent_vec = np.zeros(50) # as word vectors are of zero length
    weight_sum =0; # num of words with a valid vector in the sentence/r
eview
    for word in sent: # for each word in a review/sentence
        if word in w2v_words and word in tfidf_feat:
            vec = w2v_model.wv[word]
#             tf_idf = tf_idf_matrix[row, tfidf_feat.index(word)]
            # to reduce the computation we are
            # dictionary[word] = idf value of word in whole courpus
            # sent.count(word) = tf valeus of word in this review
            tf_idf = dictionary[word]*(sent.count(word)/len(sent))
            sent_vec += (vec * tf_idf)
            weight_sum += tf_idf
    if weight_sum != 0:
        sent_vec /= weight_sum
    sent_vectors_test.append(sent_vec)
    row += 1
```

```
100%|██████████| 33000/33000 [06:00<00:00, 91.57it/s]
```

```
(33000, 50)
```

(33000, 50)

In [0]:
```
tfidf_sent_vectors_test= np.array(sent_vectors_test)
print(tfidf_sent_vectors_test.shape)
```

(33000, 50)

In [0]:
```
#removing Test columns from original dataframe
X_train=X_train.drop(['text'],axis=1)
X_test=X_test.drop(['text'],axis=1)
```

In [0]:
```
#checking shapes of X_train and tfidf_sent_vectors_train
print(X_train.shape)
print(tfidf_sent_vectors_train.shape)#there are 50 features we got afte
r vectorization
```

(67000, 26)
(67000, 50)

In [0]:
```
#stacking both previous features and vectorized features
X_train=hstack((X_train,tfidf_sent_vectors_train))
X_test= hstack((X_test,tfidf_sent_vectors_test))
```

In [0]:
```
#shape after hstacking
print(X_train.shape)
print(X_test.shape)
```

(67000, 76)
(33000, 76)

In [0]:
```
scale = StandardScaler(with_mean=False)
X_train = scale.fit_transform(X_train)
X_test = scale.transform(X_test)
```

In [0]:
```
#Below is the funcction for cross validation using randomsearch cv
#This function takes algorithm and data and takes paramteres and return
```

```
s the best parameteres
def xgboost_cv(algorithm,X_train,Y_train):
  random_search = RandomizedSearchCV(algorithm, param_distributions=par
ams, cv=2, verbose=1,scoring='neg_log_loss',n_jobs=-1)
  result=random_search.fit(X_train, Y_train)
  return result
```

In [0]:

```
import xgboost as xgb
from xgboost import XGBClassifier
from sklearn.model_selection import RandomizedSearchCV
params = {'learning_rate' : np.arange(0.1,1,0.1),'max_depth': [3, 4, 5
],'n_estimators': np.arange(100,500,100)}
xgb = XGBClassifier(objective='binary:logistic',eval_metric='logloss',s
ilent=True)
result=xgboost_cv(xgb,X_train,y_train)
```

```
Fitting 2 folds for each of 10 candidates, totalling 20 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 12.8min finished
```

In [0]:

```
print(result.best_params_)
```

```
{'n_estimators': 300, 'max_depth': 3, 'learning_rate': 0.2}
```

In [0]:

```
tuned_learn_rate=result.best_params_['learning_rate']
tuned_n_estimator =result.best_params_['n_estimators']
tuned_depth=result.best_params_['max_depth']
```

In [0]:

```
xgb =XGBClassifier(objective='binary:logistic',eval_metric='logloss',si
lent=True,learning_rate=tuned_learn_rate,max_depth=tuned_depth,n_estima
tors=tuned_n_estimator)
model=xgb.fit(X_train,y_train)
predict_y=model.predict_proba(X_test)
predicted_y =np.argmax(predict_y,axis=1)
```

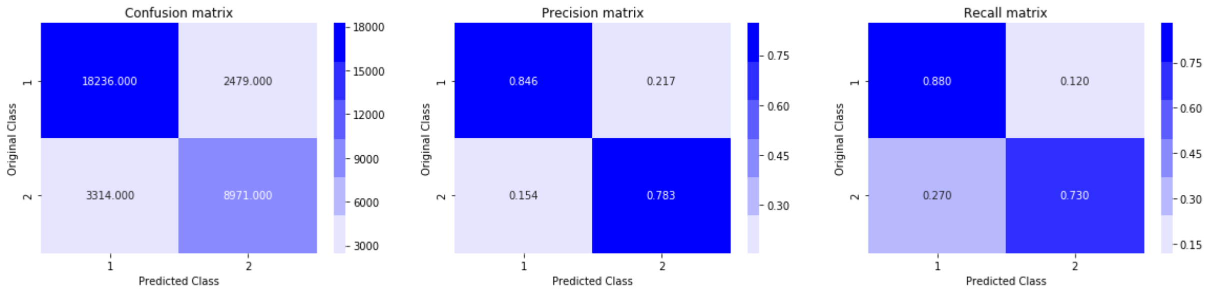
In [0]:

```
print(log_loss(y_test,predict_y))
```

```
0.349310788291553
```

In [0]:
```
y_test = list(map(int, y_test.values))
```

In [0]:
```
plot_confusion_matrix(y_test, predicted_y)
```



Pretty table representation

In [0]:
```
pip install beautifultable
```

```
Collecting beautifultable
  Downloading https://files.pythonhosted.org/packages/d9/56/eaf1b9f2b32
3e05dce573f88c72eaa0107610db709b8bee97b776903ac55/beautifultable-0.8.0-
py2.py3-none-any.whl
Installing collected packages: beautifultable
Successfully installed beautifultable-0.8.0
```

In [0]:
```
from beautifultable import BeautifulTable
table = BeautifulTable()
table.column_headers = ["model",'Vectorization',"log-loss"]
table.append_row(["Logistic regresion",'GLOVE',0.520035530431])
table.append_row(["Linear SVM",'GLOVE',0.489669093534])
table.append_row(["XGBOOST",'GLOVE', 0.357054433715])
table.append_row(["Logistic regresion",'TFIDF',0.4433669178735429])
table.append_row(["Linear SVM",'TFIDF',0.6602315214861881])
```

```python
table.append_row(["XGBOOST_TUNED",'TFIDFW2V', 0.349310788291553])
print(table)
```

```
+---------------------+----------------+-----------+
|        model        | Vectorization | log-loss |
+---------------------+----------------+-----------+
| Logistic regresion |     GLOVE      |   0.52    |
+---------------------+----------------+-----------+
|     Linear SVM      |     GLOVE      |   0.49    |
+---------------------+----------------+-----------+
|       XGBOOST       |     GLOVE      |  0.357    |
+---------------------+----------------+-----------+
| Logistic regresion |     TFIDF      |  0.443    |
+---------------------+----------------+-----------+
|     Linear SVM      |     TFIDF      |   0.66    |
+---------------------+----------------+-----------+
|    XGBOOST_TUNED    |    TFIDFW2V    |  0.349    |
+---------------------+----------------+-----------+
```

In [0]: