

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

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

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 and storing into dataframe

In [2]:

```
tfidf_tr=pd.read_csv('X_tr.csv',index_col='id')
tfidf_test=pd.read_csv('X_test.csv',index_col='id')
```

In [2]:

```
#http://www.sqlitetutorial.net/sqlite-python/create-tables/
def create_connection(db_file):
    """ create a database connection to the SQLite database
        specified by db_file
    :param db_file: database file
    :return: Connection object or None
```

```
def create_connection(db_file):
```

```
    """
    try:
        conn = sqlite3.connect(db_file)
        return conn
    except Error as e:
        print(e)

    return None
```

```
def checkTableExists(dbcon):
    cursr = dbcon.cursor()
    str = "select name from sqlite_master where type='table'"
    table_names = cursr.execute(str)
    print("Tables in the database:")
    tables = table_names.fetchall()
    print(tables[0][0])
    return len(tables)
```

In [3]:

```
read_db = 'w2v_data.db'
conn_r = create_connection(read_db)
checkTableExists(conn_r)
conn_r.close()
```

Tables in the database:
X_tr

In [4]:

```
# try to sample data according to the computing power you have
if os.path.isfile(read_db):
    conn_r = create_connection(read_db)
    if conn_r is not None:
        # for selecting first 1M rows
        # data = pd.read_sql_query("""SELECT * FROM data LIMIT 100001;""", conn_r)

        # for selecting random points
        X_tr = pd.read_sql_query("SELECT * From X_tr ;", conn_r, index_col='index')
        X_test = pd.read_sql_query("SELECT * From X_test ;", conn_r, index_col='index')
        y_tr = pd.read_sql_query("SELECT * From y_tr ;", conn_r, index_col='index')
        y_test = pd.read_sql_query("SELECT * From y_test ;", conn_r, index_col='index')
        conn_r.commit()
        conn_r.close()
```

In [6]:

```
print("Number of data points in w2v train data :", X_tr.shape)
print("Number of data points in w2v test data :", X_test.shape)
print("Number of data points in tfidf train data :", tfidf_tr.shape)
print("Number of data points in tfidf test data :", tfidf_test.shape)
print("Number of data points in y train data :", y_tr.shape)
print("Number of data points in y test data :", y_test.shape)
```

Number of data points in w2v train data : (70000, 795)
Number of data points in w2v test data : (30000, 795)
Number of data points in tfidf train data : (70000, 10431)
Number of data points in tfidf test data : (30000, 10431)
Number of data points in y train data : (70000, 1)
Number of data points in y test data : (30000, 1)

In [5]:

```
X_tr.drop(['id'], axis=1, inplace=True)
X_test.drop(['id'], axis=1, inplace=True)
tfidf_tr.drop(['Unnamed: 0'], axis=1, inplace=True)
tfidf_test.drop(['Unnamed: 0'], axis=1, inplace=True)
```

In [6]:


```

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 [11]:

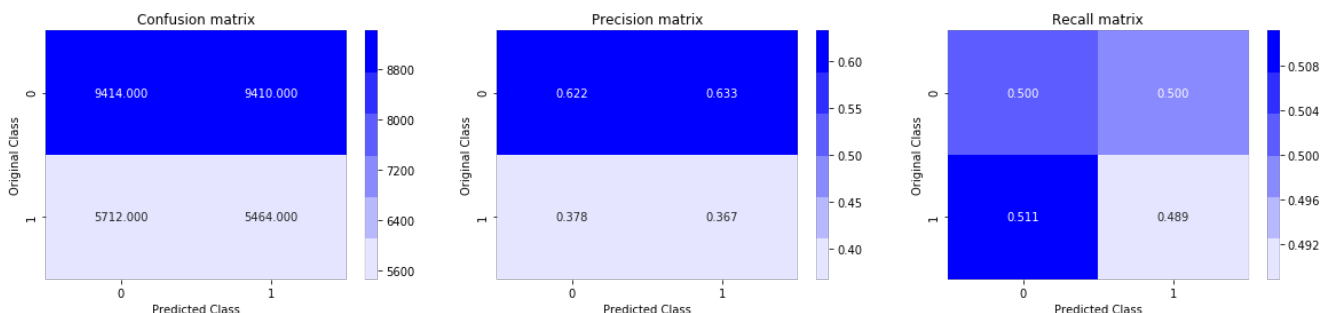
```

# we need to generate 9 numbers and the sum of numbers should be 1
# one solution is to generate 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 0.8903648083856596



4.4 Logistic Regression with hyperparameter tuning

5. Assignments

1. Try out models (Logistic regression, Linear-SVM) with simple TF-IDF vectors instead of TD_IDF weighted word2Vec.
2. Hyperparameter tune XgBoost using RandomSearch to reduce the log-loss.

In [12]:

```

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_iter=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, class_weight='balanced',
n_jobs=-1)
    clf.fit(tfidf_tr, y_tr)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(tfidf_tr, y_tr)
    predict_y = sig_clf.predict_proba(tfidf_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, class_weight='balanced', n_jobs=-1)
clf.fit(tfidf_tr, y_tr)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(tfidf_tr, y_tr)

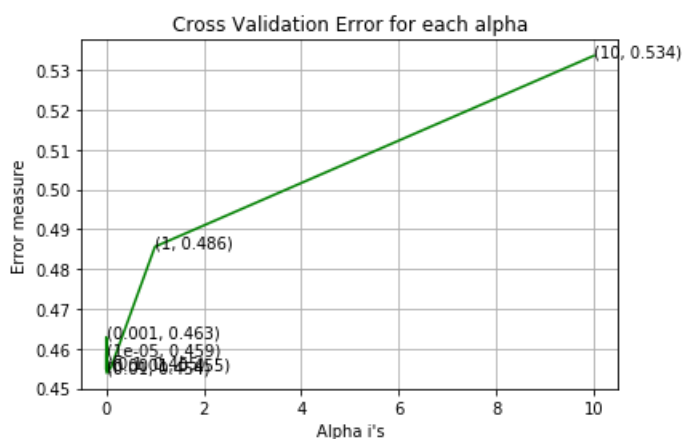
predict_y = sig_clf.predict_proba(tfidf_tr)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_tr, pr
edict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(tfidf_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log 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: 0.4586103513769618
For values of alpha = 0.0001 The log loss is: 0.45470105873123134
For values of alpha = 0.001 The log loss is: 0.4628862357629175
For values of alpha = 0.01 The log loss is: 0.4539122229943025
For values of alpha = 0.1 The log loss is: 0.4553349506769389
For values of alpha = 1 The log loss is: 0.48559136848346973
For values of alpha = 10 The log loss is: 0.5335445830523713

```

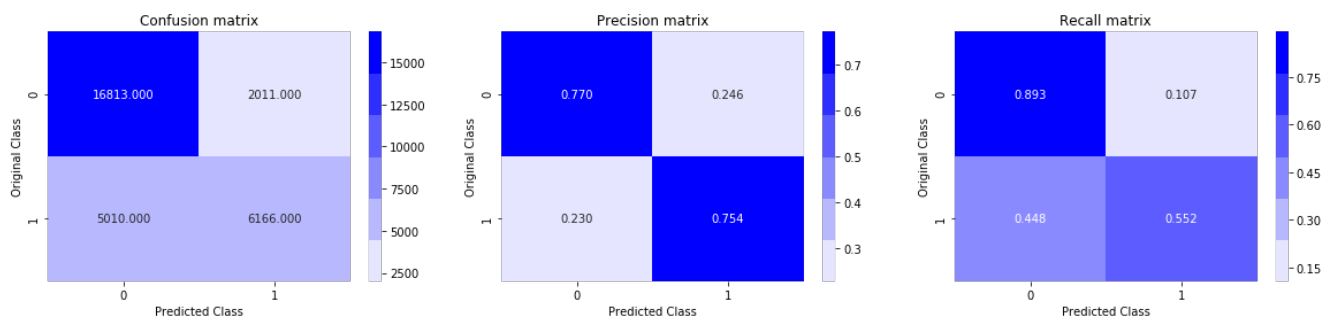


```

For values of best alpha = 0.01 The train log loss is: 0.4515707506374777
For values of best alpha = 0.01 The test log loss is: 0.4539122229943025

```

For values of best alpha = 0.01 the test log loss is: 0.19912225919929
 Total number of data points : 30000



4.5 Linear SVM with hyperparameter tuning

In [13]:

```
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_iter=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, class_weight='balanced', n_jobs=-1)
    clf.fit(tfidf_tr, y_tr)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(tfidf_tr, y_tr)
    predict_y = sig_clf.predict_proba(tfidf_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.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', n_jobs=-1)
clf.fit(tfidf_tr, y_tr)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(tfidf_tr, y_tr)

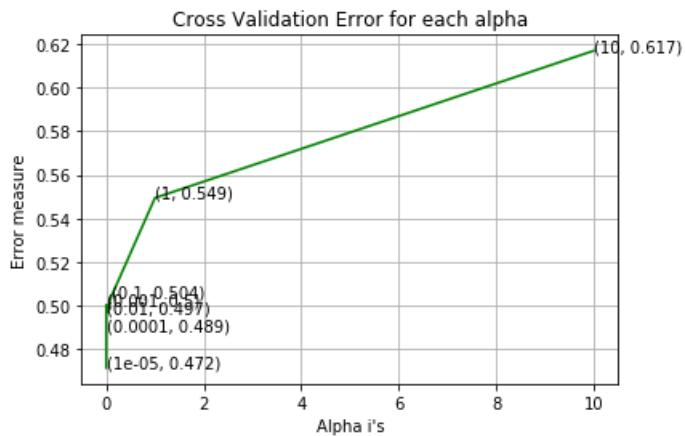
predict_y = sig_clf.predict_proba(tfidf_tr)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:", log_loss(y_tr, predict_y, labels=clf.classes_, eps=1e-15))
predict_y = sig_clf.predict_proba(tfidf_test)
print('For values of best alpha = ', alpha[best_alpha], "The test log loss is:", log_loss(y_test, predict_y, labels=clf.classes_, eps=1e-15))
```

```

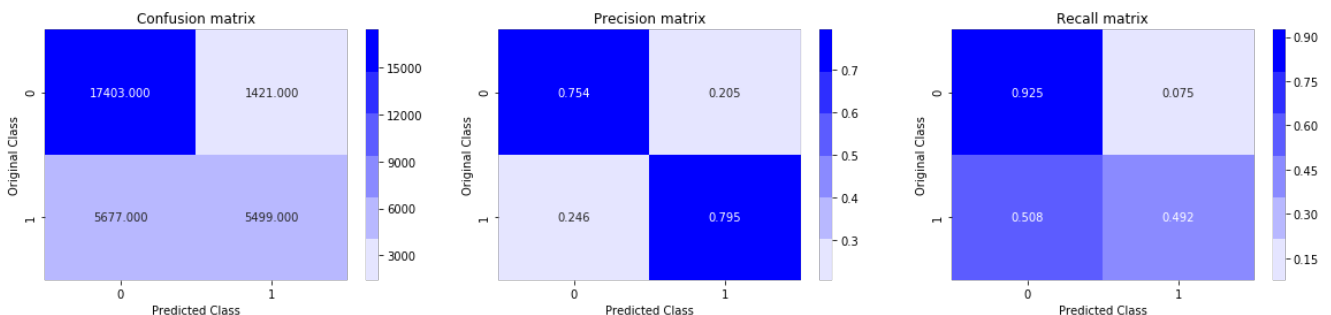
predict_y, labels=clf.classes_, eps=1e-15))
predicted_y = np.argmax(predict_y,axis=1)
print("Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)

```

For values of alpha = 1e-05 The log loss is: 0.4715146993871238
 For values of alpha = 0.0001 The log loss is: 0.4888994376355062
 For values of alpha = 0.001 The log loss is: 0.5003194764724606
 For values of alpha = 0.01 The log loss is: 0.4970081428108785
 For values of alpha = 0.1 The log loss is: 0.50397998259496
 For values of alpha = 1 The log loss is: 0.549369036541818
 For values of alpha = 10 The log loss is: 0.6168077207631271



For values of best alpha = 1e-05 The train log loss is: 0.46860359813016933
 For values of best alpha = 1e-05 The test log loss is: 0.4715146993871238
 Total number of data points : 30000



In [9]:

```

#log_error_array=[]
#for i in parameters[0]['n_estimators']:
#    for j in parameters[0]['max_depth']:
#        xgb_model = xgb.XGBClassifier(class_weight='balanced',n_jobs=-
1,max_depth=j,n_estimators=i)
#        xgb_model.fit(X_tr, y_tr.values.ravel())
#        predict_y = xgb_model.predict_proba(X_test)
#        log_error_array.append(log_loss(y_test, predict_y, labels=xgb_model.classes_, eps=1e-15))
#        print('For values of n_estimators = ', i, ' and max_depth=',j, "The log loss
is:",log_loss(y_test, predict_y, labels=xgb_model.classes_, eps=1e-15))

```

4.6 XGBoost

In [9]:

```

xgb_model = xgb.XGBClassifier(class_weight='balanced',n_jobs=-1)
param_grid = {
    'max_depth': [5,6,7,8],
    'n_estimators': [20,30,50,100]}
rand_search = RandomizedSearchCV(xgb_model, param_grid, cv=5,n_jobs=-1,random_state=42,n_iter=4)

print("Randomized search..")
search time start = time.time()

```

```

rand_search.fit(X_tr, y_tr)
print("Randomized search time:", time.time() - search_time_start)

best_score = rand_search.best_score_
best_params = rand_search.best_params_
print("Best score: {}".format(best_score))
print("Best params: ")
for param_name in sorted(best_params.keys()):
    print('%s: %r' % (param_name, best_params[param_name]))

```

Randomized search..
 Randomized search time: 1703.1051998138428
 Best score: 0.8301285714285714
 Best params:
 max_depth: 8
 n_estimators: 50

In [10]:

```

xgb_model =
xgb.XGBClassifier(max_depth=best_params['max_depth'], n_estimators=best_params['n_estimators'], class_
_weight='balanced', n_jobs=-1)
xgb_model.fit(X_tr, y_tr)
y_train_pred=[]
y_test_pred=[]
for j in range(0, X_tr.shape[0], 1000):
    y_train_pred.extend(xgb_model.predict_proba(X_tr[j:j+1000])[:,1])
print("The train log loss is:", log_loss(y_tr, y_train_pred, labels=xgb_model.classes_, eps=1e-15))
for j in range(0, X_test.shape[0], 1000):
    y_test_pred.extend(xgb_model.predict_proba(X_test[j:j+1000])[:,1])
print("The test log loss is:", log_loss(y_test, y_test_pred, labels=xgb_model.classes_, eps=1e-15))

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

```

The train log loss is: 0.27774656280167404
 The test log loss is: 0.3472660616601662
 Total number of data points : 30000

