In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
import re
import time
import warnings
from nltk.stem import PorterStemmer
from bs4 import BeautifulSoup
import re
import sqlite3
from sqlalchemy import create_engine
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

import scipy.stats as sc

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
import scipy.stats as sc
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV
from tqdm import tqdm
from gensim.models import Word2Vec
```

In [2]:
```
df = pd.read_csv('train.csv')
```

In [3]:
```
STOP_WORDS = stopwords.words("english")
def preprocess(x):
    x = str(x).lower()
    x = x.replace(",000,000","m").replace(",000","k").replace("what's","what i
s").replace("won't","will not").replace("isn't","is not")\
                            .replace("’", "'").replace("’", "'").replace("n't"
," not").replace("cannot","can not").replace("can't","can not")\
                            .replace("'ve"," have").replace("i'm","i am").repl
ace("'re"," are")\
                            .replace("he's","he is").replace("she's","she is")
.replace("'s"," own")\
                            .replace("%", " percent ").replace("₹", " rupee ")
.replace("$", " dollar ")\
                           .replace("€", " euro ").replace("'ll", " will")

    x = re.sub(r"([0-9]+)000000",r"\1m",x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)

    porter =  PorterStemmer()
    pattern = re.compile("\W")

    if type(x) == type(''):
        x = re.sub(pattern,' ',x)

    if type(x) == type(''):
        x = porter.stem(x)
        bs4 = BeautifulSoup(x)
        x = bs4.get_text()

    return x
```

In [4]:
```
df['question1'] = df['question1'].fillna('').apply(preprocess)
df['question2'] = df['question2'].fillna('').apply(preprocess)
df['question1'] = df['question1'].apply(lambda x: str(x))
df['question2'] = df['question2'].apply(lambda x: str(x))
from sklearn.feature_extraction.text import TfidfVectorizer
df.head()
```

Out[4]:

	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 [5]:
```
questions = df['question1'].values + df['question2'].values
print(questions[1])
tfidf = TfidfVectorizer(lowercase = False)
tfidf_questions = tfidf.fit_transform(questions)
question_feat = dict(zip(tfidf.get_feature_names(),list(tfidf.idf_)))
```

```
what is the story of kohinoor  koh i noor  diamond what would happen if the i
ndian government stole the kohinoor  koh i noor  diamond back
```

In [6]:
```
list_sent_question = []
for i in questions:
    list_sent_question.append(i.split())

list_sent_q1 = []
for sent in df['question1'].values:
    list_sent_q1.append(sent.split())

list_sent_q2 = []
for sent in df['question2'].values:
    list_sent_q2.append(sent.split())
```

In [7]:
```
w2v_model = Word2Vec(list_sent_question,min_count = 5,size = 384,workers = 4)
w2v_word = list(w2v_model.wv.vocab)
df = pd.read_csv('train.csv')
```

In [8]:
```
tf_idf_feat = tfidf.get_feature_names()
q1_vec = []
row = 0
for sent in tqdm(list_sent_q1,unit = 'sent',desc = 'Question1'):
    sent_vec = np.zeros(384)
    weight_sum = 0
    for word in sent:
        if word in w2v_word:
            try:
                vec = w2v_model.wv[word]
                tfidf = question_feat[word] * sent.count(word)
                sent_vec += (vec*tfidf)
                weight_sum += tfidf
            except:
                pass
    sent_vec/= weight_sum
    q1_vec.append(sent_vec)
    row += 1
df['q1_feats_m'] = list(q1_vec)
```

```
Question1: 100%|██████████████████████████████████████████████████████████|
404290/404290 [05:08<00:00, 1311.08sent/s]
```

In [9]:
```
q2_vec = []
row = 0
for sent in tqdm(list_sent_q2,unit = 'sent',desc = 'Question2'):
    sent_vec = np.zeros(384)
    weight_sum = 0
    for word in sent:
        if word in w2v_word:
            try:
                vec = w2v_model.wv[word]
                tfidf = question_feat[word] * sent.count(word)
                sent_vec += (vec*tfidf)
                weight_sum += tfidf
            except:
                pass
    sent_vec/= weight_sum
    q2_vec.append(sent_vec)
    row += 1
df['q2_feats_m'] = list(q2_vec)
```

```
Question2: 100%|██████████████████████████████████████████████████████████|
404290/404290 [04:49<00:00, 1394.72sent/s]
```

In [10]:
```
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
k")

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

In [12]:
```
print(len(q1_vec))
print(len(q2_vec))
```

```
404290
404290
```

In [13]:
```
df1 = dfnlp.drop(['qid1','qid2','question1','question2'],axis=1)
df2 = dfppro.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=
1)
df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [ ]:
```
df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)
```

In [ ]:
```
df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)
```

In [ ]:
```
df3_q1.head()
```

In [ ]:
```
df3_q2.head()
```

In [ ]:
```
if not os.path.isfile('final_features_tfidfw2v.csv'):
    df3_q1.insert(loc=0, column='id', value=np.arange(0,df3_q1.shape[0]))
    df3_q2.insert(loc=0, column='id', value=np.arange(0,df3_q2.shape[0]))
    final_df  = pd.merge(df1,df2, on='id')
    final_df  = pd.merge(final_df, df3_q1,on='id')
    final_df  = pd.merge(final_df, df3_q2,on='id')
```

In [ ]:
```
y_true = result['is_duplicate']
y_true = list(map(int, y_true.values))
result.drop(['id','is_duplicate'], axis=1, inplace=True)
result.head()
```

In [ ]:
```
# split the data into test and train by maintaining same distribution of outpu
t varaible 'y_true' [stratify=y_true]
X_train, X_test, y_train, y_test = train_test_split(result, y_true, stratify=y
_true, test_size=0.2)
```

In [ ]:
```
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[1])/test_len, "Class 1: ",int(test_distr[1])/
test_len)
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler(with_mean = False)
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=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 Stochast
ic 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_, ep
s=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre
dict_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)
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
s:",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 i
s:",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)
```

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

# read more about SGDClassifier() at http://scikit-learn.org/stable/modules/ge
nerated/sklearn.linear_model.SGDClassifier.html
# ------------------------------
# default parameters
# SGDClassifier(loss='hinge', penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_i
ntercept=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 Stochast
ic 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_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_, ep
s=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test, pre
dict_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', rando
m_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 i
s:",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 i
s:",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)
```

In [ ]:
```
params = {"learning_rate":sc.uniform(0.01,0.1),
             "n_estimators":sc.randint(10,250),
             "max_depth":sc.randint(4,10),
            }
xgb_classifier = xgb.XGBClassifier(objective = 'binary:logistic')
grid = RandomizedSearchCV(xgb_classifier, params, cv = 3, scoring = "log_loss"
, verbose = 1, random_state = 0)
grid.fit(X_train,y_train)
print(xgb_classifier.best_params_)
```

In [ ]:
```
predict_y = xgb_classifier.predict_proba(X_train)
print("The train log loss is:",log_loss(y_train, predict_y, eps=1e-15))
predict_y = xgb_classifier.predict_proba(X_test)
print("/n The test log loss is:",log_loss(y_test, predict_y, eps=1e-15))
predicted_y =np.argmax(predict_y,axis=1)
print("/n Total number of data points :", len(predicted_y))
plot_confusion_matrix(y_test, predicted_y)
```