3.6 Featurizing text data with tfidf weighted word-vectors

In [61]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import datetime
import numpy as np
from nltk.corpus import stopwords
from sklearn.preprocessing import normalize
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
warnings.filterwarnings("ignore")
import sys
from collections import Counter
from collections import Counter, defaultdict
import os
import pandas as pd
import numpy as np
from tqdm import tqdm

# exctract word2vec vectors
# https://github.com/explosion/spaCy/issues/1721
# http://landinghub.visualstudio.com/visual-cpp-build-tools
import spacy


import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
import datetime
import joblib
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
```

In [2]:

```
# avoid decoding problems
df = pd.read_csv("train.csv")

df_train = df[:70000]
df_test = df[70000:100000]

# encode questions to unicode
# https://stackoverflow.com/a/6812069
# ----------------- python 2 ---------------------
# df['question1'] = df['question1'].apply(lambda x: unicode(str(x),"utf-8"))
# df['question2'] = df['question2'].apply(lambda x: unicode(str(x),"utf-8"))
# ----------------- python 3 ---------------------
df_train['question1'] = df_train['question1'].apply(lambda x: str(x))
df_train['question2'] = df_train['question2'].apply(lambda x: str(x))

df_test['question1'] = df_test['question1'].apply(lambda x: str(x))
df_test['question2'] = df_test['question2'].apply(lambda x: str(x))
```

In [3]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

startTime3 = datetime.datetime.now()
print("Current Time = ",startTime3)

# merge texts
questions_train = list(df_train['question1']) + list(df_train['question2'])
questions_test = list(df_test['question1']) + list(df_test['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(questions_train)
tfidf.transform(questions_test)

# dict key:word and value:tf-idf score
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 14:38:05.153887
Time taken to run this cell 0:00:11.290700
```

- After we find TF-IDF scores, we convert each question to a weighted average of word2vec vectors by these scores.
- here we use a pre-trained GLOVE model which comes free with "Spacy". https://spacy.io/usage/vectors-similarity (https://spacy.io/usage/vectors-similarity)
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [7]:
```
print(np.shape(df_train['question1']))
```

```
(70000,)
```

In [8]:
```
# vectorizing train data of question1
# last comment worked: https://stackoverflow.com/questions/49964028/spacy-oserror

startTime3 = datetime.datetime.now()
print("Current Time = ",startTime3)

# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(df_train['question1']):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)

df_train['q1_feats_m_train'] = list(vecs1)

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 14:44:37.209707

100%|████████████████████████████████████████████████████████████████████████
██| 70000/70000 [55:24<00:00, 21.05it/s]

Time taken to run this cell 0:55:27.805032
```

In [10]:
```
# vectorizing train data of question2

startTime3 = datetime.datetime.now()
print("Current Time = ",startTime3)

vecs2 = []
for qu2 in tqdm(df_train['question2']):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc1), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)

df_train['q2_feats_m_train'] = list(vecs2)

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 15:41:58.087603

100%|██████████████████████████████████████████████████████████████████████
██| 70000/70000 [54:53<00:00, 21.26it/s]

Time taken to run this cell 0:54:53.656920
```

In [11]:
```
# vectorizing test data of question1

startTime3 = datetime.datetime.now()
print("Current Time = ",startTime3)

# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
for qu1 in tqdm(df_test['question1']):
    doc1 = nlp(qu1)
    # 384 is the number of dimensions of vectors
    mean_vec1 = np.zeros([len(doc1), len(doc1[0].vector)])
    for word1 in doc1:
        # word2vec
        vec1 = word1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word1)]
        except:
            idf = 0
        # compute final vec
        mean_vec1 += vec1 * idf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)

df_test['q1_feats_m_test'] = list(vecs1)

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 16:36:51.773923

100%|██████████████████████████████████████████████████████████████████████
██| 30000/30000 [29:00<00:00, 14.91it/s]

Time taken to run this cell 0:29:03.005378
```

In [12]:
```
# vectorizing test data of question2

startTime3 = datetime.datetime.now()
print("Current Time = ",startTime3)

vecs2 = []
for qu2 in tqdm(df_test['question2']):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc1), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word2)]
        except:
            #print word
            idf = 0
        # compute final vec
        mean_vec2 += vec2 * idf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)

df_test['q2_feats_m_test'] = list(vecs2)

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 17:05:54.822112

100%|████████████████████████████████████████████████████████████████
██| 30000/30000 [28:58<00:00, 17.25it/s]

Time taken to run this cell 0:28:59.068028
```

In [29]:
```
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

ques_1_train_df = pd.DataFrame(df_train['q1_feats_m_train'].tolist())
ques_1_test_df = pd.DataFrame(df_train['q2_feats_m_train'].tolist())

ques_2_train_df = pd.DataFrame(df_test['q1_feats_m_test'].tolist())
ques_2_test_df = pd.DataFrame(df_test['q2_feats_m_test'].tolist())

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 17:43:40.710343
Time taken to run this cell 0:01:31.432278
```

In [13]:
```
#prepro_features_train.csv (Simple Preprocessing Feartures)
#nlp_features_train.csv (NLP Features)
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-
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previ
```

In [14]:
```
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 [17]:
```
# dataframe of nlp features
df1.head()
```

Out[17]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first
0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	

In [20]:
```
df1_train = df1[:70000]
df1_test = df1[70000:100000]
df1_test[:10]
```

Out[20]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_
70000	70000	1	0.666644	0.666644	0.999967	0.599988	0.833319	0.624992	
70001	70001	0	0.399992	0.249997	0.000000	0.000000	0.153845	0.133332	
70002	70002	0	0.666644	0.499988	0.666644	0.499988	0.571420	0.571420	
70003	70003	0	0.999967	0.749981	0.999975	0.799984	0.999986	0.777769	
70004	70004	0	0.249997	0.249997	0.142855	0.083333	0.187499	0.124999	
70005	70005	0	0.499988	0.499988	0.499975	0.199996	0.499992	0.333330	
70006	70006	1	0.999967	0.749981	0.666644	0.666644	0.833319	0.714276	
70007	70007	1	0.666656	0.571420	0.749981	0.599988	0.699993	0.583328	
70008	70008	0	0.000000	0.000000	0.666644	0.249997	0.249997	0.142856	
70009	70009	0	0.999967	0.999967	0.599988	0.374995	0.749991	0.461535	

In [21]:
```
# data before preprocessing
df2.head()
```

Out[21]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	w
0	0	1	1	66	57	14	12	10.0	23.0	
1	1	4	1	51	88	8	13	4.0	20.0	
2	2	1	1	73	59	14	10	4.0	24.0	
3	3	1	1	50	65	11	9	0.0	19.0	
4	4	3	1	76	39	13	7	2.0	20.0	

In [22]:
```
df2_train = df2[:70000]
df2_test = df2[70000:100000]
df2_test[:10]
```

Out[22]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_
70000	70000	1	1	40	33	8	6	4.0	
70001	70001	1	1	77	76	12	14	1.0	
70002	70002	1	1	44	38	7	7	4.0	
70003	70003	1	1	45	35	8	6	5.0	
70004	70004	4	1	105	116	15	22	3.0	
70005	70005	1	1	47	38	9	6	2.0	
70006	70006	2	1	40	34	7	6	4.0	
70007	70007	1	1	56	69	10	12	7.0	
70008	70008	1	1	46	75	8	14	2.0	
70009	70009	1	1	54	40	12	8	6.0	

In [24]:
```
# dataframe of nlp features
df3.head()
```

Out[24]:

	id
0	0
1	1
2	2
3	3
4	4

In [25]:
```
df3_train = df3[:70000]
df3_test = df3[70000:100000]
df3_test[:10]
```

Out[25]:

	id
70000	70000
70001	70001
70002	70002
70003	70003
70004	70004
70005	70005
70006	70006
70007	70007
70008	70008
70009	70009

In [30]:
```
df3_q1_train = pd.DataFrame(ques_1_train_df, index= df3_train.index)
df3_q1_test = pd.DataFrame(ques_1_test_df, index= df3_test.index)

df3_q2_train = pd.DataFrame(ques_2_train_df, index= df3_train.index)
df3_q2_test = pd.DataFrame(ques_2_test_df, index= df3_test.index)
```

In [32]:
```
print("Number of features in nlp dataframe :", df1.shape[1])
print("Number of features in preprocessed dataframe :", df2.shape[1])
print("Number of features in question1 w2v  dataframe :", ques_1_train_df.shape[1
print("Number of features in question2 w2v  dataframe :", ques_1_test_df.shape[1]
print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+ques_
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 384
Number of features in question2 w2v  dataframe : 384
Number of features in final dataframe  : 797
```

In [33]:
```
# storing the final features of train data to csv file
startTime3 = datetime.datetime.now()
print("Current Time = ",startTime3)

if not os.path.isfile('final_features_train_w2v.csv'):
    df3_q1_train['id']=df1_train['id']
    df3_q2_train['id']=df1_train['id']
    df1_train  = df1_train.merge(df2_train, on='id',how='left')
    df2_train  = df3_q1_train.merge(df3_q2_train, on='id',how='left')
    result  = df1_train.merge(df2_train, on='id',how='left')
    result.to_csv('final_features_train_w2v.csv')

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 17:47:45.417730
Time taken to run this cell 0:06:12.196241
```

In [34]:
```
final_features_train_w2v = pd.read_csv("final_features_train_w2v.csv")
final_features_train_w2v[:5]
```

Out[34]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_w
0	0	0	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	
1	1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	
2	2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	
3	3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
4	4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	

5 rows × 797 columns

In [35]:
```
# storing the final features of test data to csv file
startTime = datetime.datetime.now()
print("Current Time = ",startTime)

if not os.path.isfile('final_features_test_w2v.csv'):
    df3_q1_test['id']=df1_test['id']
    df3_q2_test['id']=df1_test['id']
    df1_test  = df1_test.merge(df2_test, on='id',how='left')
    df2_test  = df3_q1_test.merge(df3_q2_test, on='id',how='left')
    result_test  = df1_test.merge(df2_test, on='id',how='left')
    result_test.to_csv('final_features_test_w2v.csv')

print("Time taken to run this cell {}".format(datetime.datetime.now() - startTime
```

```
Current Time =  2019-05-16 17:58:22.846895
Time taken to run this cell 0:01:15.298492
```

In [36]:
```
final_features_test_w2v = pd.read_csv("final_features_test_w2v.csv")
final_features_test_w2v[:5]
```

Out[36]:

	Unnamed: 0	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	las
0	0	70000	1	0.666644	0.666644	0.999967	0.599988	0.833319	0.624992	
1	1	70001	0	0.399992	0.249997	0.000000	0.000000	0.153845	0.133332	
2	2	70002	0	0.666644	0.499988	0.666644	0.499988	0.571420	0.571420	
3	3	70003	0	0.999967	0.749981	0.999975	0.799984	0.999986	0.777769	
4	4	70004	0	0.249997	0.249997	0.142855	0.083333	0.187499	0.124999	

5 rows × 797 columns

In [37]:
```
# remove the first row
start = datetime.datetime.now()
print("Current Time = ",start)

final_features_train_w2v.drop(final_features_train_w2v.index[0], inplace=True)
y_true_train = final_features_train_w2v['is_duplicate']
final_features_train_w2v.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplac

final_features_test_w2v.drop(final_features_test_w2v.index[0], inplace=True)
y_true_test = final_features_test_w2v['is_duplicate']
final_features_test_w2v.drop(['Unnamed: 0', 'id','is_duplicate'], axis=1, inplace

current_time = datetime.datetime.now()
print("Time taken to run this cell: ",current_time-start)
```

```
Current Time =  2019-05-16 17:59:46.116073
Time taken to run this cell:  0:00:03.002798
```

In [38]:
```
final_features_train_w2v.head()
```

Out[38]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_le
1	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	
2	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	
3	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	
4	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	
5	0.666656	0.571420	0.888879	0.799992	0.705878	0.705878	1.0	0.0	

5 rows × 794 columns

In [39]:
```
final_features_test_w2v.head()
```

Out[39]:

	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_le
1	0.399992	0.249997	0.000000	0.000000	0.153845	0.133332	0.0	0.0	
2	0.666644	0.499988	0.666644	0.499988	0.571420	0.571420	0.0	0.0	
3	0.999967	0.749981	0.999975	0.799984	0.999986	0.777769	0.0	1.0	
4	0.249997	0.249997	0.142855	0.083333	0.187499	0.124999	0.0	0.0	
5	0.499988	0.499988	0.499975	0.199996	0.499992	0.333330	0.0	0.0	

5 rows × 794 columns

In [ ]:
```
# resetting index
def reset_index(data_frame):
    data_frame = data_frame.reset_index()
    data_frame['index_col'] = data_frame.index

    data_frame = data_frame.drop("index", axis=1)
    data_frame = data_frame.drop("index_col", axis=1)
    return(data_frame)
```

In [ ]:
```
final_features_train_w2v = reset_index(final_features_train_w2v)
final_features_test_w2v = reset_index(final_features_test_w2v)
```

In [40]:
```
# https://stackoverflow.com/questions/7368789/convert-all-strings-in-a-list-to-in
y_true_train = list(map(int, y_true_train.values))
y_true_test = list(map(int, y_true_test.values))
```

In [41]:
```
print(np.shape(y_true_train))
print(np.shape(y_true_test))
```

```
(69999,)
(29999,)
```

Converting strings to numerics

In [42]:
```
# after we read from sql table each entry was read it as a string
# we convert all the features into numaric before we apply any model

start = datetime.datetime.now()
print("Current Time = ",start)

cols = list(final_features_train_w2v.columns)
for i in cols:
    final_features_train_w2v[i] = final_features_train_w2v[i].apply(pd.to_numeric
    print(i)

current_time = datetime.datetime.now()
print("Time taken to run this cell: ",current_time-start)

start2 = datetime.datetime.now()
print("Current Time = ",start2)

cols = list(final_features_test_w2v.columns)
for i in cols:
    final_features_test_w2v[i] = final_features_test_w2v[i].apply(pd.to_numeric)
    print(i)

current_time = datetime.datetime.now()
print("Time taken to run this cell: ",current_time-start2)
```

```
383_y
Time taken to run this cell:  0:15:25.797741
Current Time =  2019-05-16 18:17:18.437590
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
```

In [44]:
```
X_train_w2v = final_features_train_w2v
X_test_w2v = final_features_test_w2v

y_train_w2v = y_true_train
y_test_w2v = y_true_test

print("Number of data points in train data :",X_train_w2v.shape)
print("Number of data points in test data :",X_test_w2v.shape)

import joblib                                          #  * DO NOT RUN *
joblib.dump(X_train_w2v,"X_train_w2v.pkl")
joblib.dump(X_test_w2v,"X_test_w2v.pkl")
joblib.dump(y_train_w2v,"y_train_w2v.pkl")
joblib.dump(y_test_w2v,"y_test_w2v.pkl")
```

```
Number of data points in train data : (69999, 794)
Number of data points in test data : (29999, 794)
```

Out[44]:
```
['y_test_w2v.pkl']
```

In [ ]:
```
#Loading the saved Train data frame
X_train_w2v = joblib.load("X_train_w2v.pkl")
X_test_w2v = joblib.load("X_test_w2v.pkl")

y_train_w2v = joblib.load("y_train_w2v.pkl")
y_test_w2v = joblib.load("y_test_w2v.pkl")
```

In [45]:
```
# Standardizing the data
from sklearn.preprocessing import StandardScaler
std_scal = StandardScaler(with_mean=False)
std_scal.fit(X_train_w2v)
X_train_w2v = std_scal.transform(X_train_w2v)
X_test_w2v = std_scal.transform(X_test_w2v)
```

In [49]:
```
print("-"*10, "Distribution of output variable in train data", "-"*10)
train_distr = Counter(y_train_w2v)
train_len = len(y_train_w2v)
print("Class 0: ",int(train_distr[0])/train_len,"Class 1: ", int(train_distr[1])/
print("-"*10, "Distribution of output variable in train data", "-"*10)
test_distr = Counter(y_test_w2v)
test_len = len(y_test_w2v)
print("Class 0: ",int(test_distr[1])/test_len, "Class 1: ",int(test_distr[1])/tes
```

```
---------- Distribution of output variable in train data ----------
Class 0:  0.6275375362505179 Class 1:  0.3724624637494821
---------- Distribution of output variable in train data ----------
Class 0:  0.3727124237474582 Class 1:  0.3727124237474582
```

In [50]:
```
# 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 p

    A =(((C.T)/(C.sum(axis=1))).T)
    #divid each element of the confusion matrix with the sum of elements in that

    # 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
    # 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 c
    # C = [[1, 2],
    #     [3, 4]]
    # C.sum(axis = 0)  axis=0 corresonds to columns and axis=1 corresponds to rows
    # 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, yticklab
    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, yticklab
    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, yticklab
    plt.xlabel('Predicted Class')
    plt.ylabel('Original Class')
    plt.title("Recall matrix")

    plt.show()
```

In [51]:
```
print(np.shape(X_train_w2v))
print(np.shape(y_train_w2v))
```

```
(69999, 794)
(69999,)
```

In [52]:
```
# checking for NaN values
def NaN_values(data_frame):
    bool_series = pd.isnull(data_frame)

    # displayind data only with team = NaN
    print("Number of rows with NaN values = ",len(data_frame[bool_series]))
    return (data_frame[bool_series][:10])
```

In [53]:
```
x = NaN_values(X_train_w2v)
x
```

```
Number of rows with NaN values =  15360000
```

Out[53]:
```
array([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan])
```

In [54]:
```
X_train_w2v = np.nan_to_num(X_train_w2v)   #Not required if there are no Null
```

In [55]:
```
x = NaN_values(X_train_w2v)
x
```

```
Number of rows with NaN values =  0
```

Out[55]:
```
array([], dtype=float64)
```

In [56]:
```
x = NaN_values(X_test_w2v)
x
```

```
Number of rows with NaN values =  23039232
```

Out[56]:
```
array([nan, nan, nan, nan, nan, nan, nan, nan, nan, nan])
```

In [57]:
```
X_test_w2v = np.nan_to_num(X_test_w2v)
x = NaN_values(X_test_w2v)
x
```

```
Number of rows with NaN values =  0
```

Out[57]:
```
array([], dtype=float64)
```

Hyperparameter tuning for XGBoost

In [59]:
```
# https://towardsdatascience.com/doing-xgboost-hyper-parameter-tuning-the-smart-w
# https://www.kaggle.com/tilii7/hyperparameter-grid-search-with-xgboost
# https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xg
# https://www.kaggle.com/phunter/xgboost-with-gridsearchcv

def xgboost_fun(X_train,y_train,X_test,y_test):
    startTime3 = datetime.datetime.now()
    print("Current Time = ",startTime3)

    from xgboost import XGBClassifier
    xgb = XGBClassifier()

    params = {
              'objective':['binary:logistic'],
              'learning_rate': [0.05, 0.10, 0.15],
              'max_depth': [3, 4, 5]}

    grid = GridSearchCV(estimator=xgb, param_grid=params, scoring="neg_log_loss")
    grid.fit(X_train, y_train)
    print('\n All results:')
    print(grid.cv_results_)
    print('\n Best estimator:')
    print(grid.best_estimator_)
    print('\n Best score:')
    print(grid.best_score_)
    print('\n Best parameters:')
    print(grid.best_params_)

    best_learning_rate = grid.best_params_['learning_rate']
    print("Best learning rate: ", best_learning_rate)

    best_max_depth = grid.best_params_['max_depth']
    print("Best learning rate: ", best_max_depth)

    import xgboost as xgb
    startTime = datetime.datetime.now()
    print("Current Time = ",startTime)

    params = {}
    params['objective'] = 'binary:logistic'
    params['eval_metric'] = 'logloss'
    params['eta'] = best_learning_rate
    params['max_depth'] = best_max_depth

    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, 400, watchlist, early_stopping_rounds=20, ve

    xgdmat = xgb.DMatrix(X_train,y_train)
    predict_y = bst.predict(d_test)
    print("The test log loss is:",log_loss(y_test, predict_y))

    predicted_y =np.array(predict_y>0.5,dtype=int)
```

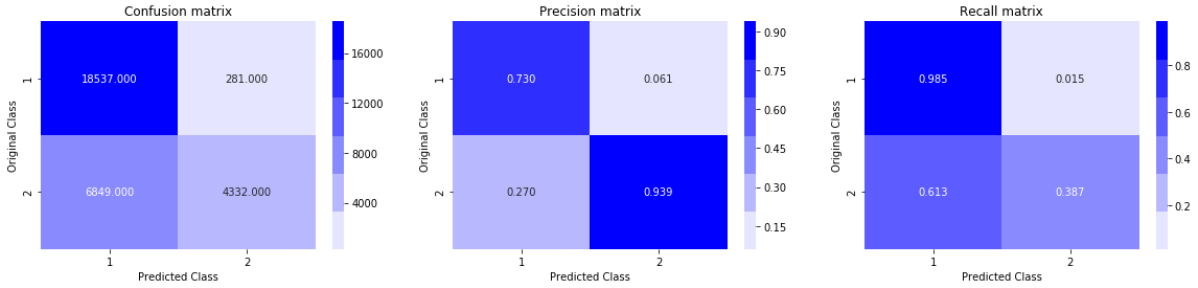
```
    print("Total number of data points :", len(predicted_y))
    plot_confusion_matrix(y_test, predicted_y)

    current_time = datetime.datetime.now()
    print("Time taken to run this cell: ",current_time-startTime3)
```

In [62]:
```
xgboost_fun(X_train_w2v,y_train_w2v,X_test_w2v,y_test_w2v)
```

```
[01:56:39] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 62 extra
nodes, 0 pruned nodes, max_depth=5
[01:56:56] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 46 extra
nodes, 0 pruned nodes, max_depth=5
Stopping. Best iteration:
[26]    train-logloss:0.357027  valid-logloss:0.38383

The test log loss is: 0.45605028405095477
Total number of data points : 29999
```




```
Time taken to run this cell:  7:25:41.483273
```

In [ ]: