3.6 Featurizing text data with tfidf weighted word-vectors

In [4]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
import time
import warnings
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
import os
import pandas as pd
import numpy as np
from tqdm import tqdm
```

In [2]:
```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)


id1='1gTfCTD3fz-3NJnfYLm59nZFN3WC3fzfD'
downloaded1 = drive.CreateFile({'id': id1})
downloaded1.GetContentFile('df_fe_without_preprocessing_train.csv')

id2='1JncN1Fyt-ND_yZXOzqEfcRsYMTKqtu7Q'
downloaded1 = drive.CreateFile({'id': id2})
downloaded1.GetContentFile('nlp_features_train.csv')


id3='10QDGTSI5PEV9e7CTpfzsXRpUwRIsJA-J'
downloaded1 = drive.CreateFile({'id': id3})
downloaded1.GetContentFile('train.csv')
```

```
     |████████████████████████████████| 993kB 45.3MB/s eta 0:00:01
  Building wheel for PyDrive (setup.py) ... done
```

In [ ]:
```
# avoid decoding problems
df = pd.read_csv("train.csv")
from sklearn.model_selection import train_test_split
# merge texts
df_train,df_test=train_test_split(df,test_size=0.2)

# 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 [4]:
```
df.head(2)
```

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

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(df_train['question1']) + list(df_train['question2'])

tfidf = TfidfVectorizer(lowercase=False, )
tfidf.fit_transform(questions)

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

- 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)](https://spacy.io/usage/vectors-similarity)

- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [6]:
```
# 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(list(df_train['question1'])):
    doc1 = nlp(qu1)
    # 96 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:
            tfidf = word2tfidf[str(word1)] * (qu1.count(str(word1))/len(qu1.split()))
        except:
            tfidf = 0
        # compute final vec
        mean_vec1 += vec1 * tfidf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df_train['q1_feats_m'] = list(vecs1)
```

```
100%|██████████| 323432/323432 [43:06<00:00, 125.03it/s]
```

In [ ]:
```
df_train.to_csv('mid1.csv')
df_test.to_csv('test.csv')
```

In [11]:
```
vecs2 = []
from tqdm import tqdm
for qu2 in tqdm(list(df_train['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score

        try:

            tfidf = word2tfidf[str(word2)] * (qu2.count(str(word2))/len(qu2.split()))
        except:
            #print word
            tfidf = 0
        # compute final vec

        #print(tfidf)
        mean_vec2 += vec2 * tfidf
    mean_vec2 = mean_vec2.mean(axis=0)

    vecs2.append(mean_vec2)
df_train['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 323432/323432 [41:04<00:00, 131.24it/s]
```

In [12]:
```
#Test Features Questions1 and Questions2
# 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(list(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:
            tfidf = word2tfidf[str(word1)] * (qu1.count(str(word1))/len(qu1.split()))
        except:
            tfidf = 0
        # compute final vec
        mean_vec1 += vec1 * tfidf
    mean_vec1 = mean_vec1.mean(axis=0)
    vecs1.append(mean_vec1)
df_test['q1_feats_m'] = list(vecs1)

########################################

vecs2 = []
for qu2 in tqdm(list(df_test['question2'])):
    doc2 = nlp(qu2)
    mean_vec2 = np.zeros([len(doc2), len(doc2[0].vector)])
    for word2 in doc2:
        # word2vec
        vec2 = word2.vector
        # fetch df score
        try:
            tfidf = word2tfidf[str(word2)] * (qu2.count(str(word2))/len(qu2.split()))
        except:
            #print word
            tfidf = 0
        # compute final vec
```

```
        mean_vec2 += vec2 * tfidf
    mean_vec2 = mean_vec2.mean(axis=0)
    vecs2.append(mean_vec2)
df_test['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 80858/80858 [10:44<00:00, 125.52it/s]
100%|██████████| 80858/80858 [10:07<00:00, 133.04it/s]
```

In [13]:
```
df_train.head(2)
```

Out[13]:

	id	qid1	qid2	question1	question2	is_duplicate	q1_feats_m	q2_feats_m
287851	287851	408683	408684	What are some examples of terrestrial animals?	What are terrestrial animals? What are example...	1	[9.50959499180317, -2.984976351261139, -11.809...	[4.78945130109787, -5.7303591668605804, -9.730...
172952	172952	266924	266925	Why is ASEAN one of the most peaceful and pros...	Why and how did Lebanon become the most peacef...	0	[0.6344003081321716, -1.813245631987229, -12.8...	[-7.4249771372415125, -6.347789332270622, -11....

In [ ]:
```
#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-1')
else:
    print("download df_fe_without_preprocessing_train.csv from drive or run previous notebook")
```

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


df3_train = df_train.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df3_test = df_test.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)

df3_q1_train = pd.DataFrame(df3_train.q1_feats_m.values.tolist(), index= df3_train.index)
df3_q2_train = pd.DataFrame(df3_train.q2_feats_m.values.tolist(), index= df3_train.index)

df3_q1_test = pd.DataFrame(df3_test.q1_feats_m.values.tolist(), index= df3_test.index)
df3_q2_test = pd.DataFrame(df3_test.q2_feats_m.values.tolist(), index= df3_test.index)
```

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

Out[55]:

	id	is_duplicate	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	0	0.999980	0.833319	0.999983	0.999983	0.916659	0.785709	0.0	1.0	2.0	13.0	10(
1	1	0	0.799984	0.399996	0.749981	0.599988	0.699993	0.466664	0.0	1.0	5.0	12.5	8(
2	2	0	0.399992	0.333328	0.399992	0.249997	0.399996	0.285712	0.0	1.0	4.0	12.0	6(
3	3	0	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0	0.0	2.0	12.0	3(
4	4	0	0.399992	0.199998	0.999950	0.666644	0.571420	0.307690	0.0	1.0	6.0	10.0	6:

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

Out[56]:

	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
0	0	1	1	66	57	14	12	10.0	23.0	0.434783	2	0
1	1	4	1	51	88	8	13	4.0	20.0	0.200000	5	3
2	2	1	1	73	59	14	10	4.0	24.0	0.166667	2	0
3	3	1	1	50	65	11	9	0.0	19.0	0.000000	2	0
4	4	3	1	76	39	13	7	2.0	20.0	0.100000	4	2

In [16]:
```
# Questions 1 tfidf weighted word2vec
df3_q1_train.head()
```

Out[16]:

	0	1	2	3	4	5	6	7	8	9	10	11	1
287851	9.509595	-2.984976	-11.809498	-9.398076	3.067511	8.801890	18.589524	13.412171	-8.947324	3.094738	1.691513	8.095941	-6.30305
172952	0.634400	-1.813246	-12.816122	-12.080643	5.457196	6.322567	15.836552	13.633682	0.312183	8.396769	0.150165	7.360641	-5.96279
247768	4.431505	-4.361650	-14.430818	-18.104730	-2.806155	6.768903	9.910724	4.264563	-0.473978	10.360899	-4.802288	-1.079249	-19.05542
97009	8.108846	-5.319327	-6.192122	-12.733797	-1.011927	4.555990	10.343346	8.802996	-1.492674	6.124089	-2.084184	3.179069	-8.06257
223297	3.949012	-6.848152	-4.856686	-11.317435	-9.334562	1.511885	7.551820	10.383319	0.304397	9.664529	-6.060358	3.826666	-12.00161

5 rows × 96 columns

In [17]:
```
df3_q1_test.head()
```

Out[17]:

	0	1	2	3	4	5	6	7	8	9	10	11	
157561	25.026054	-12.854203	-9.678908	-11.264161	-10.215891	-2.118202	31.566062	5.966860	-6.212604	12.947480	-4.721270	-6.075376	-8.
55473	3.153236	-9.807701	-4.287981	-10.497754	-5.162055	-4.743260	8.219638	2.725274	0.194639	8.444565	-8.259800	8.336918	-18.
58292	26.724261	-34.266608	-16.617415	-27.571456	-6.003354	5.019149	37.134820	13.056351	-15.587229	10.477277	2.765593	-14.594760	-7.
124205	5.418724	-5.579635	-5.298835	-6.650882	-0.867129	2.277497	4.982583	14.247606	-7.309681	6.909354	-3.446842	-7.683994	-5.
211135	-0.915871	-4.670748	-7.094173	-7.577088	-2.944890	4.100913	15.464195	8.516277	3.587404	1.071980	2.100014	5.814831	-8.

5 rows × 96 columns

In [ ]:
```
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 :", df3_q1.shape[1])
print("Number of features in question2 w2v  dataframe :", df3_q2.shape[1])
print("Number of features in final dataframe  :", df1.shape[1]+df2.shape[1]+df3_q1.shape[1]+df3_q2.shape[1])
```

In [ ]:
```
# storing the final features to csv file
if not os.path.isfile('final_features.csv'):
    df3_q1_train['id']=df_train['id']
    df3_q2_train['id']=df_train['id']

    df3_q1_test['id']=df_test['id']
    df3_q2_test['id']=df_test['id']

    df1  = df1.merge(df2, on='id',how='inner')

    df2_train  = df3_q1_train.merge(df3_q2_train, on='id',how='inner')
    df2_test   = df3_q1_test.merge(df3_q2_test, on='id',how='inner')

    result_train  = df1.merge(df2_train, on='id',how='inner')
    result_test  = df1.merge(df2_test, on='id',how='inner')


    result_train.to_csv('final_features_train.csv')
    result_test.to_csv('final_features_test.csv')
```

In [19]:
```
result_test.shape,result_train.shape
```

Out[19]:
```
((80858, 220), (323432, 220))
```

XGBoost on TFIDF Weighted W2V on Questions

In [1]:
```
!wget --header="Host: doc-00-58-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x6
```

```
--2019-08-13 19:14:42--  https://doc-00-58-docs.googleusercontent.com/docs/securesc/qgbh6kjdsd98f3gihp270udf3ooq8huc/72qp0q01ae1bdm21jllpmu1i3u8nh8fa/1565719200000/15923203954827925545/15923203954827925545/1NE36Qhqn27kUkhKpra_P2f3-tXsYl9xG?e=download (https://doc-00-58-docs.googleusercontent.com/docs/securesc/qgbh6kjdsd98f3gihp270udf3ooq8huc/72qp0q01ae1bdm21jllpmu1i3u8nh8fa/1565719200000/15923203954827925545/15923203954827925545/1NE36Qhqn27kUkhKpra_P2f3-tXsYl9xG?e=download)
Resolving doc-00-58-docs.googleusercontent.com (doc-00-58-docs.googleusercontent.com)... 74.125.197.132, 2607:f8b0:400e:c03::84
Connecting to doc-00-58-docs.googleusercontent.com (doc-00-58-docs.googleusercontent.com)|74.125.197.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/csv]
Saving to: ‘final_features_train.csv’

final_features_trai     [     <=>            ]   1.16G   160MB/s    in 8.9s

2019-08-13 19:14:51 (133 MB/s) - ‘final_features_train.csv’ saved [1244544516]
```

In [2]:

```
!wget --header="Host: doc-08-58-docs.googleusercontent.com" --header="User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x6
```

```
--2019-08-13 19:15:20--  https://doc-08-58-docs.googleusercontent.com/docs/securesc/qgbh6kjdsd98f3gihp270udf3ooq8huc/3f4htdnq5pk7ck92fiukbahq2efnmbdj/1565719200000/15923203954827925545/15923203954827925545/11kPWHuyFUvomF-vwmkkIiKXryZNfXJIY?e=download (https://doc-08-58-docs.googleusercontent.com/docs/securesc/qgbh6kjdsd98f3gihp270udf3ooq8huc/3f4htdnq5pk7ck92fiukbahq2efnmbdj/1565719200000/15923203954827925545/15923203954827925545/11kPWHuyFUvomF-vwmkkIiKXryZNfXJIY?e=download)
Resolving doc-08-58-docs.googleusercontent.com (doc-08-58-docs.googleusercontent.com)... 74.125.197.132, 2607:f8b0:400e:c03::84
Connecting to doc-08-58-docs.googleusercontent.com (doc-08-58-docs.googleusercontent.com)|74.125.197.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/csv]
Saving to: ‘final_features_test.csv’

final_features_test     [           <=>      ] 296.67M   103MB/s    in 2.9s

2019-08-13 19:15:23 (103 MB/s) - ‘final_features_test.csv’ saved [311078704]
```

In [5]:

```
result_train=pd.read_csv('final_features_train.csv')
result_test=pd.read_csv('final_features_test.csv')
```

In [ ]:

In [6]:
```
import xgboost as xgb
from sklearn.model_selection import RandomizedSearchCV

y_train=result_train['is_duplicate']
y_test=result_test['is_duplicate']
X_tr=result_train.drop(['is_duplicate'],axis=1)
X_te=result_test.drop(['is_duplicate'],axis=1)


x_cfl=xgb.XGBClassifier()

prams = {
    'learning_rate':[0.01,0.1,0.2],
     'n_estimators':[100,500,1000],
     'max_depth':[3,5,8],
    'colsample_bytree':[0.1,0.5,1],
    'subsample':[0.1,0.5,1]
}
random_cfl1=RandomizedSearchCV(x_cfl,param_distributions=prams,scoring='neg_log_loss',cv=2,verbose=1,n_jobs=-1)
random_cfl1.fit(X_tr,y_train)
```

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

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  20 out of  20 | elapsed: 104.6min finished
```

Out[6]:
```
RandomizedSearchCV(cv=2, error_score='raise-deprecating',
                   estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                           colsample_bylevel=1,
                                           colsample_bynode=1,
                                           colsample_bytree=1, gamma=0,
                                           learning_rate=0.1, max_delta_step=0,
                                           max_depth=3, min_child_weight=1,
                                           missing=None, n_estimators=100,
                                           n_jobs=1, nthread=None,
                                           objective='binary:logistic',
                                           random_state=0, reg_alpha=0,
                                           reg_lambda=1, scale_pos_weight=1,
                                           seed=None, silent=None, subsample=1,
                                           verbosity=1),
                   iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'colsample_bytree': [0.1, 0.5, 1],
```

```
                                        'learning_rate': [0.01, 0.1, 0.2],
                                        'max_depth': [3, 5, 8],
                                        'n_estimators': [100, 500, 1000],
                                        'subsample': [0.1, 0.5, 1]},
                    pre_dispatch='2*n_jobs', random_state=None, refit=True,
                    return_train_score=False, scoring='neg_log_loss', verbose=1)
```

In [ ]:

```
import shutil
import os
source = os.listdir("/content")
destination = "/content/drive/My Drive"
for files in source:
    if files.endswith(".csv"):
        shutil.copy(files,destination)
```

In [7]:

```
print (random_cfl1.best_params_)
```

```
{'subsample': 0.1, 'n_estimators': 500, 'colsample_bytree': 0.1, 'max_depth': 5, 'learning_rate': 0.01}
```

In [8]:

```
from sklearn.calibration import CalibratedClassifierCV

cfl=xgb.XGBClassifier(n_estimators=500,subsample=0.1,learning_rate=0.01,colsample_bytree=0.1,max_depth=5)
cfl.fit(X_tr,y_train)
c_cfl=CalibratedClassifierCV(x_cfl,method='sigmoid')
c_cfl.fit(X_tr,y_train)

predict_y = c_cfl.predict_proba(X_tr)
```

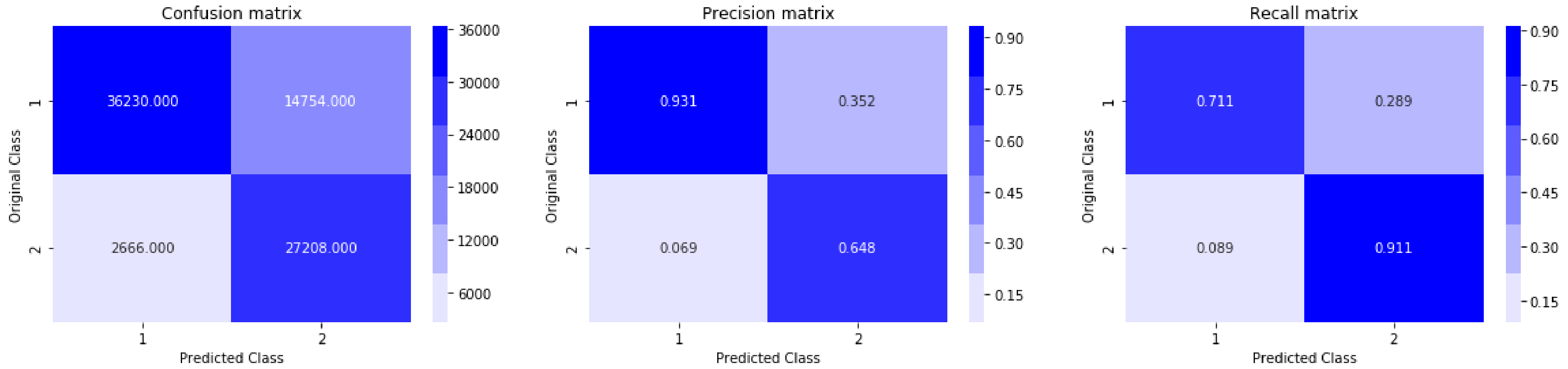
In [11]:

```

print ('train loss',log_loss(y_train, predict_y))
predict_y = c_cfl.predict_proba(X_te)
print ('test loss',log_loss(y_test, predict_y))
```

```
train loss 0.39909822481038637
test loss 0.40790524897003333
```

In [14]:
```
#y_test = list(map(int, y_test.values))
from sklearn.metrics import confusion_matrix
import seaborn as sns
predicted_y =np.array(predict_y>0.5,dtype=int)
plot_confusion_matrix(y_test, predicted_y[:,1])
```



In [10]:
```
# This function plots the confusion matrices given y_i, y_i_hat.
from sklearn.metrics import log_loss
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()
```

Conclusion

So the Train Logloss is 0.39909822481038637
and the Test Logloss 0.40790524897003333

In [ ]:

```
```