3.6 Featurizing text data with tfidf weighted word-vectors

In [1]:

```
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

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

In [2]:

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

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

In [3]:

```
df.head()
```

Out[3]:

	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 [4]:

```
# Take 100k datasample
df = df.sample(n=100000, random_state=1)
df.shape
```

Out[4]:

```
(100000, 6)
```

In [5]:

```
# Split the data into 70,30 train and test data
from sklearn.model_selection import train_test_split
tr, ts = train_test_split(df, test_size=0.3, random_state=1, stratify=df['is_duplicate'].values)
```

In [6]:

```
tr.shape, ts.shape
```

Out[6]:

```
((70000, 6), (30000, 6))
```

In [7]:

```
tr.head()
```

Out[7]:

	id	qid1	qid2	question1	question2	is_duplicate
149483	149483	235456	235457	How do I translate in Android?	How would you translate "螳螂捕蝉, 黄雀在后"?	0
146085	146085	179014	230839	Can a pet bird be trained to live without a ca...	How do airports keep birds away?	0
337094	337094	44878	204218	What is the best way to teach a child how to s...	How do you teach your kid to swim?	1
115033	115033	187657	187658	How do I add a location to my business page on...	Can a Facebook Page check-in to a Place? Or, w...	0
190104	190104	289081	289082	What purpose did the Roman Colosseum have?	What purpose does the Colosseum serve?	1

In [8]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
# merge texts
questions = list(tr['question1']) + list(tr['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
- It is trained on Wikipedia and therefore, it is stronger in terms of word semantics.

In [9]:

```
# en_vectors_web_lg, which includes over 1 million unique vectors.
nlp = spacy.load('en_core_web_sm-2.2.0/en_core_web_sm/en_core_web_sm-2.2.0')

vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
# For 'question1' feature train data
for qu1 in tqdm(list(tr['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)

tr['q1_feats_m'] = list(vecs1)
```

```
100%|| 70000/70000 [07:11<00
:00, 162.19it/s]
```

In [10]:

```
# For 'question2' feature train data
vecs2 = []
for qu2 in tqdm(list(tr['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:
            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)

tr['q2_feats_m'] = list(vecs2)
```

```
100%|| 70000/70000 [07:12<00
:00, 161.69it/s]
```

In [11]:

```
vecs1 = []
# https://github.com/noamraph/tqdm
# tqdm is used to print the progress bar
# For 'question1' feature train data
for qu1 in tqdm(list(ts['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)

ts['q1_feats_m'] = list(vecs1)
```

```
100%|| 30000/30000 [03:05<00
:00, 161.72it/s]
```

In [12]:

```
# For 'question2' feature train data
vecs2 = []
for qu2 in tqdm(list(ts['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:
            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)

ts['q2_feats_m'] = list(vecs2)
```

```
100%|██████████| 30000/30000 [03:06<00
:00, 161.01it/s]
```

In [13]:

```
tr.shape, ts.shape
```

Out[13]:

```
((70000, 8), (30000, 8))
```

In [14]:

```
#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_basicfe_train.csv'):
    dfppro = pd.read_csv("df_basicfe_train.csv",encoding='latin-1')
else:
    print("download df_basicfe_train.csv from drive or run previous notebook")
```

In [15]:

```
# Take 100k and split in same proportion
# Random state parameter give the good idea that if you give random state to any value, and recompile a
gain and again,
# the result will always the same.

df1 = dfnlp.sample(n=100000, random_state=1)
df2 = dfppro.sample(n=100000, random_state=1)
df1.shape, df2.shape
```

Out[15]:

```
((100000, 21), (100000, 17))
```

In [16]:

```
df_tr1, df_ts1 = train_test_split(df1, test_size=0.3, random_state=1, stratify=df1['is_duplicate'].valu
es)
df_tr2, df_ts2 = train_test_split(df2, test_size=0.3, random_state=1, stratify=df2['is_duplicate'].valu
es)
df_tr1.shape, df_tr2.shape, df_ts1.shape, df_ts2.shape
```

Out[16]:

```
((70000, 21), (70000, 17), (30000, 21), (30000, 17))
```

In [17]:

```
df_tr1 = df_tr1.drop(['qid1','qid2','question1','question2'],axis=1)
df_ts1 = df_ts1.drop(['qid1','qid2','question1','question2'],axis=1)
df_tr2 = df_tr2.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
df_ts2 = df_ts2.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)
```

In [28]:

```
# Just take tfidf w2v feature only and remove others
df3 = tr.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)

# Store tfidf w2v of question1 train data
df3_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)

# Store tfidf w2v of question2 train data
df3_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)

# Just take tfidf w2v feature only and remove others
df3 = ts.drop(['qid1','qid2','question1','question2','is_duplicate'],axis=1)

# Store tfidf w2v of question1 train data
df4_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index= df3.index)

# Store tfidf w2v of question2 train data
df4_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index= df3.index)
```

In [29]:

```
# dataframe of advance nlp feature of train data
df_tr1.head()
```

Out[29]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_dif
149483	149483	0	0.499975	0.249994	0.499975	0.249994	0.333328	0.333328	0.0	1.0	0.0
146085	146085	0	0.249994	0.124998	0.000000	0.000000	0.166664	0.066666	1.0	0.0	9.0
337094	337094	1	0.666644	0.399992	0.399992	0.333328	0.499994	0.333331	1.0	0.0	4.0
115033	115033	0	0.599988	0.374995	0.285710	0.222220	0.416663	0.238094	0.0	0.0	9.0
190104	190104	1	0.666644	0.666644	0.666644	0.499988	0.666656	0.571420	0.0	1.0	1.0

In [30]:

```
# dataframe of advance nlp feature of test data
df_ts1.head()
```

Out[30]:

	id	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	ctc_min	ctc_max	last_word_eq	first_word_eq	abs_len_dif
303972	303972	0	0.833319	0.555549	0.999986	0.636358	0.857137	0.599997	0.0	0.0	6.0
72206	72206	1	0.499992	0.499992	0.333322	0.199996	0.363633	0.333331	1.0	0.0	1.0
106335	106335	1	0.999983	0.857131	0.999986	0.874989	0.999992	0.866661	0.0	1.0	2.0
268194	268194	0	0.583328	0.538457	0.636358	0.583328	0.499998	0.466665	0.0	0.0	2.0
33364	33364	0	0.333322	0.166664	0.000000	0.000000	0.166664	0.090908	0.0	0.0	5.0

In [31]:

```
# Dataframe of basic feature of train data
df_tr2.head()
```

Out[31]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	t
149483	149483	1	1	30	37	6	6	2.0	12.0	0.166667	2	
146085	146085	2	1	66	32	15	6	1.0	20.0	0.050000	3	
337094	337094	3	4	50	34	12	8	4.0	19.0	0.210526	7	
115033	115033	1	1	56	96	12	19	4.0	27.0	0.148148	2	
190104	190104	1	1	42	38	7	6	4.0	13.0	0.307692	2	

In [32]:

```
# Dataframe of basic feature of train data
df_ts2.head()
```

Out[32]:

	id	freq_qid1	freq_qid2	q1len	q2len	q1_n_words	q2_n_words	word_Common	word_Total	word_share	freq_q1+q2	t
303972	303972	1	1	60	102	13	19	11.0	31.0	0.354839	2	
72206	72206	3	2	61	59	10	11	3.0	19.0	0.157895	5	
106335	106335	1	1	79	68	15	12	10.0	27.0	0.370370	2	
268194	268194	1	1	133	140	26	30	11.0	49.0	0.224490	2	
33364	33364	1	1	32	58	6	11	1.0	17.0	0.058824	2	

In [33]:

```
# Questions 1 tfidf weighted word2vec
df3_q1.head()
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	...
149483	-35.970820	15.233681	-83.945030	1.821328	33.741198	44.038382	-10.648363	-7.947617	49.464797	-49.422297	...
146085	-85.020419	46.710926	-125.288743	-75.608168	64.469008	115.124894	20.742659	158.109579	-102.035777	-23.369245	...
337094	-33.065192	-20.178773	-77.886197	-120.081859	43.659595	49.504050	-0.365680	7.494594	-37.327102	-67.218828	...
115033	-72.944806	-51.253230	-80.234965	-67.835021	104.177971	58.757157	-34.426265	13.135962	35.042460	-3.935220	...
190104	-29.736965	56.128536	-25.453947	-74.479012	-19.861213	40.580078	-90.739224	58.638960	4.982964	-60.430177	...

5 rows × 96 columns

In [35]:

```
df4_q1.head()
```

Out[35]:

	0	1	2	3	4	5	6	7	8	9	...

303972	-35.086460	30.876545	-80.416682	106.096214	47.108916	29.962705	74.211906	76.222057	-52.817488	26.348005	...
72206	1.685434	-36.735866	-196.028876	-140.657811	-43.790299	-9.156165	6.992758	-19.499774	-33.655322	-12.672375	...
106335	-111.489742	22.646898	-196.952070	-150.557910	126.045632	76.895359	-19.473369	73.001146	-117.053125	-64.015255	...
268194	-41.903405	-199.639366	-153.416367	-153.148267	-19.763226	99.206988	8.039918	34.153714	-109.584684	23.843086	...
33364	-76.174544	-46.639612	-59.099957	-51.663911	27.144306	25.683623	-2.553842	-23.118165	-23.407722	-58.685860	...

5 rows × 96 columns

In [34]:

```
# Questions 2 tfidf weighted word2vec
df3_q2.head()
```

Out[34]:

	0	1	2	3	4	5	6	7	8	9	...
149483	-33.364869	-5.495585	-87.122270	-43.383327	43.409313	8.358389	-9.309176	37.756822	41.297043	-72.184926	...
146085	-1.638967	39.372217	-87.395557	-33.371412	18.659480	44.381700	15.707307	36.774686	-23.186097	-87.426189	...
337094	-39.340501	-53.392284	-44.010859	-74.687828	43.536204	60.482706	-26.103947	-34.727356	46.506368	-86.787620	...
115033	-122.364968	35.770705	-87.295173	-171.742449	127.268932	139.834739	-8.106615	30.278763	-150.290404	-143.008661	...
190104	-9.906318	17.387377	-17.303450	-41.914522	41.149178	47.610587	-85.934912	74.996205	-38.504456	-89.565776	...

5 rows × 96 columns

In [36]:

```
df4_q2.head()
```

Out[36]:

	0	1	2	3	4	5	6	7	8	9	...
303972	-94.364372	82.517343	-207.351414	-123.893591	85.709126	88.650925	-86.536333	154.028366	-12.134686	-10.402775	...
72206	0.680685	-65.595772	-17.650448	-48.919356	70.625238	111.967245	-40.600397	-7.394346	-7.762665	-91.553182	...
106335	-117.099960	33.590637	-171.654092	-91.793823	134.984977	95.909769	0.911680	68.959064	-109.523601	-50.480547	...
268194	58.616156	-77.750569	-253.661700	-201.832817	109.530422	183.692623	18.180422	199.613442	-59.008171	-41.821395	...
33364	-77.772708	-12.111855	-121.301187	-31.978529	73.371403	92.359309	-14.790487	17.243972	-18.429706	32.780807	...

5 rows × 96 columns

In [25]:

```
print("Number of features in nlp dataframe :", df_tr1.shape[1])
print("Number of features in preprocessed dataframe :", df_tr2.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  :", df_tr1.shape[1]+df_tr2.shape[1]+df3_q1.shape[1]+df3_q
2.shape[1])
```

```
Number of features in nlp dataframe : 17
Number of features in preprocessed dataframe : 12
Number of features in question1 w2v  dataframe : 96
Number of features in question2 w2v  dataframe : 96
Number of features in final dataframe  : 221
```

In [27]:

```
# storing the final features to csv file
if not os.path.isfile('tr_finalfeatures_tfidf_w2v.csv'):
    # Assign 'id' attribute astfidf-w2v vector dataframe same as nlp or basic dataframe
    # Please observe above dataframe of basic,nlp and tfidfw2v features, you will find 'id' are all sam
e
    df3_q1['id']=df_tr1['id']
    df3_q2['id']=df_tr1['id']

    # Merge the train basic and nlp feature
    df1  = df_tr1.merge(df_tr2, on='id',how='left')
    print('Total df1 features: {0}'.format(df1.shape))

    # Merge the train tfidf-w2v question1 and question2
    df2  = df3_q1.merge(df3_q2, on='id',how='left')
    print('Total df2 features: {0}'.format(df2.shape))

    # Merge above two dataframe
    result  = df1.merge(df2, on='id',how='left')

    print('Total features: {0}'.format(result.shape))
    result.to_csv('tr_finalfeatures_tfidf_w2v.csv')
```

```
Total df1 features: (70000, 28)
Total df2 features: (70000, 193)
Total features: (70000, 220)
```

In [37]:

```
# storing the final features to csv file
if not os.path.isfile('ts_finalfeatures_tfidf_w2v.csv'):
    # Assign 'id' attribute astfidf-w2v vector dataframe same as nlp or basic dataframe
    # Please observe above dataframe of basic,nlp and tfidfw2v features, you will find 'id' are all sam
e
    df4_q1['id']=df_ts1['id']
    df4_q2['id']=df_ts1['id']

    # Merge the test basic and nlp feature
    df1  = df_ts1.merge(df_ts2, on='id',how='left')
    print('Total df1 features: {0}'.format(df1.shape))

    # Merge the test tfidf-w2v question1 and question2
    df2  = df4_q1.merge(df4_q2, on='id',how='left')
    print('Total df2 features: {0}'.format(df2.shape))

    # Merge above two dataframe
    result  = df1.merge(df2, on='id',how='left')
    print('Total features: {0}'.format(result.shape))
    result.to_csv('ts_finalfeatures_tfidf_w2v.csv')
```

```
Total df1 features: (30000, 28)
Total df2 features: (30000, 193)
Total features: (30000, 220)
```

In [ ]:

```
```