QQP3 - Featurizing text data with TF-IDF weighted Average Word2Vec

April 27, 2018

0.1 3.6 Featurizing text data with TF-IDF weighted word-vectors and Avg.word-vectors

```
In [1]: # We will only operate on the first 100k points as 8GB RAM is not enough.

        # Library imports:
        import numpy as np
        import pandas as pd
        from time import time
        import warnings
        warnings.filterwarnings('ignore')
        import matplotlib.pyplot as plt
        import sys
        import os
        from tqdm import tqdm

        # We can extract word2vec vectors using spacy
        # If there are any dependency issues, please folow these links:
        # https://github.com/explosion/spaCy/issues/1721
        # http://landinghub.visualstudio.com/visual-cpp-build-tools
        import spacy
```

```
In [2]: # We will use the following function to time our code:
        def time_taken(start_time):
            print("~> Time taken:",
                 round(time()-start_time, 2), "seconds")
            return
```

```
In [ ]: # We will import more libraries as and when required.
```

```
In [5]: st = time()

        # Import sample from the original dataset:
        df = pd.read_csv("../train/train.csv", nrows=100000)

        # Encode all the questions to unicode format:
        df['question1'] = df['question1'].apply(lambda x: str(x))
        df['question2'] = df['question2'].apply(lambda x: str(x))

        time_taken(st)
        df.shape

~> Time taken: 0.42 seconds


Out[5]: (100000, 6)
```

```
In [6]: df.head(2)
```

```
Out[6]:    id  qid1  qid2                                          question1  \
        0   0     1     2  What is the step by step guide to invest in sh...
        1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...

                                                   question2  is_duplicate
        0  What is the step by step guide to invest in sh...             0
        1  What would happen if the Indian government sto...             0
```

0.1.1 3.6.1 Computing TF-IDF weighted Average Word2Vec Vectors

```
In [7]: from sklearn.feature_extraction.text import TfidfVectorizer

        # Merge texts into a single list:
        questions = list(df['question1']) + list(df['question2'])

        # Create TfidfVectorizer instance:
        tfidf = TfidfVectorizer(lowercase=False)

        # Get the parameters in tfidf instance:
        print(tfidf)

TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
        dtype=<class 'numpy.int64'>, encoding='utf-8', input='content',
        lowercase=False, max_df=1.0, max_features=None, min_df=1,
        ngram_range=(1, 1), norm='l2', preprocessor=None, smooth_idf=True,
        stop_words=None, strip_accents=None, sublinear_tf=False,
        token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
        vocabulary=None)
```

```
In [8]: st =time()

        # Now apply tfidf transform:
        tfidf.fit_transform(questions)

        time_taken(st)
```

```
~> Time taken: 3.13 seconds
```

```
In [9]: st = time()

        # We will now take all the tf-idf vectored values into a dictionary:
        # key:word and value:tf-idf score
        word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))

        time_taken(st)
```

```
~> Time taken: 0.08 seconds
```

- We have the TF-IDF Scores. We will now convert each question to a weighted average of word2vec vectors by using these TF-IDF Scores.
- We use a pre-trained GLoVe model which comes free with spacy library. The model itself is trained on Wikipedia data. More about the pre-trained model @ https://spacy.io/usage/vectors-similarity
- Because it is trained on Wikipedia, the model is strong in terms of word semantics

```
In [ ]: # We can either load and use 'en_vectors_web_lg' or 'en_core_web_sm'.
        # The difference between the two is that, 'en_core_web_sm' is a smaller
        # model compared to 'en_vectors_web_lg'

        # If the system has less RAM, it is better to load 'en_core_web_sm' model
        # as the word-vector. The only disadvantage with a smaller model is that
        # it will give us similarity vectors with lesser accuracy.

        # Note: there are 2 full version models of word2vec models trained on
        # wikipedia data. One of them is 'en_core_web_lg' and the other one is
        # 'en_vectors_web_lg'. If we want to make changes to the word2vec model,
        # then we can load the 'en_vectors_web_lg' word2vec model, otherwise, we
        # go with 'en_core_web_lg' word2vec model.

        # Note: We can't load these pre-trained word2vec models unless and until
        # we download the models through the console of the system.

            ###############################################
            # Syntax: python -m spacy download model_name #
            ###############################################

        # Example: python -m spacy download en_core_web_lg
```

```
In [10]: # Loading the smaller model:
         nlp = spacy.load('en_core_web_sm')
         st = time()

         vector1 = []
         # https://github.com/noamraph/tqdm
         # tqdm is used to print the progress bar.
         for q1 in tqdm(list(df.question1)):
             doc = nlp(q1)
             # 384 dimensional vector
             mean_vec1 = np.zeros([len(doc), 384])
             for word in doc:
                 # word2vec:
                 vec1 = word.vector

                 # Fetch the tf-idf score:
                 try:
                     idf = word2tfidf[str(word)]
                 except:
                     idf = 0

                 # Compute final vector:
                 mean_vec1 += vec1 * idf
             mean_vec1 = mean_vec1.mean(axis=0)
             vector1.append(mean_vec1)
         df['q1_feats_m'] = list(vector1)
```

```
100%|| 100000/100000 [19:04<00:00, 87.35it/s]
```

```
In [11]: time_taken(st)
```

```
~> Time taken: 1144.88 seconds
```

```
In [12]: st = time()

         vector2 = []
         for q2 in tqdm(list(df.question2)):
             doc = nlp(q2)
             mean_vec2 = np.zeros([len(doc), 384])
             for word in doc:
                 # word2vec:
                 vec2 = word.vector
                 # Fetch idf score:
                 try:
                     idf = word2tfidf[str(word)]
                 except:
                     idf = 0
                 # Compute final vector:
                 mean_vec2 += vec2 * idf
             mean_vec2 = mean_vec2.mean(axis=0)
             vector2.append(mean_vec2)
         df['q2_feats_m'] = list(vector2)

         time_taken(st)
```

```
100%|| 100000/100000 [19:29<00:00, 85.47it/s]


~> Time taken: 1170.01 seconds
```

```
In [13]: # Get the files in the current working directory
         files_in_cwd = os.listdir()
         index = 0
         print("<idx>. <Filename>")
         for f in files_in_cwd:
             print("{}. {}".format(index, f))
             index += 1
```

```
<idx>. <Filename>
0. .ipynb_checkpoints
1. df_fe_without_preprocessing_train.csv
2. nlp_features_train.csv
3. QQP1 - BasicEDA, TextPreprocessing, BasicFeaturization, AdvancedFeaturization (NLP & Fuzzy
4. QQP1.py
5. QQP2 - Word Cloud, PCA & t-SNE 2D and 3D Visualizations of Engineered Features.ipynb
6. QQP2.py
7. QQP3 - Featurizing text data with TF-IDF weighted Average Word2Vec.ipynb
8. quora.png
9. train_dup_question_pairs.txt
10. train_non_dup_question_pairs.txt
```

```
In [14]: # We will read some previously saved .csv files like:
         # 1. nlp_features_train.csv
         # 2. df_fe_without_preprocessing_train.csv
         # and we will use the data generated now and finally merge all of the data into
         # a single pandas dataframe. The dataframe size may get really large.

         st = time()
         # Load the nlp_features_train.csv file into a dataframe:
         if os.path.isfile('nlp_features_train.csv'):
             df_nlp = pd.read_csv("nlp_features_train.csv", encoding='latin-1', nrows=100000)
         else:
             print('Generate the file by running the code in QQP1.')

         # Load the df_fe_without_preprocessing_train.csv file into a dataframe:
         if os.path.isfile('df_fe_without_preprocessing_train.csv'):
             df_pre = pd.read_csv('df_fe_without_preprocessing_train.csv', encoding=\
                                 'latin-1', nrows=100000)
```

```
        else:
            print('Generate the file by running the code in QQP1.')

        time_taken(st)

~> Time taken: 1.55 seconds
```

```
In [15]: print(df_nlp.shape)
         print(df_pre.shape)
         # We will drop the unnecessary features and only keep the required ones:
         df1 = df_nlp.drop(['qid1', 'qid2', 'question1', 'question2'], axis=1)

         # df1 corresponds to advanced nlp and fuzzy engineered features:
         df1.head()

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

```
Out[15]:    id  is_duplicate   cwc_min   cwc_max   csc_min   csc_max   ctc_min  \
         0   0             0  0.999980  0.833319  0.999983  0.999983  0.916659
         1   1             0  0.799984  0.399996  0.749981  0.599988  0.699993
         2   2             0  0.399992  0.333328  0.399992  0.249997  0.399996
         3   3             0  0.000000  0.000000  0.000000  0.000000  0.000000
         4   4             0  0.399992  0.199998  0.999950  0.666644  0.571420

             ctc_max  last_word_eq  first_word_eq  abs_len_diff  mean_len  \
         0  0.785709           0.0            1.0           0.0      13.0
         1  0.466664           0.0            1.0           0.0      12.5
         2  0.285712           0.0            1.0           0.0      12.0
         3  0.000000           0.0            0.0           0.0      12.0
         4  0.307690           0.0            1.0           0.0      10.0

            token_set_ratio  token_sort_ratio  fuzz_ratio  fuzz_parital_ratio  \
         0              100                93          93                 100
         1               86                63          66                  75
         2               63                63          43                  47
         3               28                24           9                  14
         4               67                47          35                  56

            longest_substr_ratio
         0              0.982759
         1              0.596154
         2              0.166667
         3              0.039216
         4              0.175000
```

```
In [16]: df2 = df_pre.drop(['qid1', 'qid2', 'question1', 'question2','is_duplicate'],\
                           axis=1)
```

```
         # df2 corresponds to basic engineered features:
         df2.head()

Out[16]:    id  freq_qid1  freq_qid2  q1len  q2len  q1_n_words  q2_n_words  \
         0   0          1          1     66     57          14          12
         1   1          4          1     51     88           8          13
         2   2          1          1     73     59          14          10
         3   3          1          1     50     65          11           9
         4   4          3          1     76     39          13           7

            word_Common  word_Total  word_share  freq_q1+q2  freq_q1-q2
         0         10.0        23.0    0.434783           2           0
         1          4.0        20.0    0.200000           5           3
         2          4.0        24.0    0.166667           2           0
         3          0.0        19.0    0.000000           2           0
         4          2.0        20.0    0.100000           4           2
```

```
In [17]: # our original dataset with some additional features:
         df.head()

Out[17]:    id  qid1  qid2                                          question1  \
         0   0     1     2  What is the step by step guide to invest in sh...
         1   1     3     4  What is the story of Kohinoor (Koh-i-Noor) Dia...
         2   2     5     6  How can I increase the speed of my internet co...
         3   3     7     8  Why am I mentally very lonely? How can I solve...
         4   4     9    10  Which one dissolve in water quikly sugar, salt...

                                                    question2  is_duplicate  \
         0  What is the step by step guide to invest in sh...             0
         1  What would happen if the Indian government sto...             0
         2  How can Internet speed be increased by hacking...             0
         3  Find the remainder when [math]23^{24}[/math] i...             0
         4            Which fish would survive in salt water?             0

                                                  q1_feats_m  \
         0  [122.490798712, 100.359120488, 72.0331508666, ...
         1  [-74.5846772194, 53.8620963991, 81.0885115862,...
         2  [-5.10626339912, 73.7096084356, 14.3268437684,...
         3  [5.90131050348, -34.4693912566, 48.9884575009,...
         4  [48.4207775295, 38.2941785157, 121.9611063, 54...

                                                  q2_feats_m
         0  [126.564217329, 96.0618406534, 42.2021160275, ...
         1  [-105.099983424, 79.1588504314, 77.5340879094,...
         2  [6.49532223493, 16.2452982366, 2.65493392944, ...
         3  [38.9078674316, 43.9539289773, -24.3469197154,...
         4  [31.6172962189, 62.5719087124, 1.96994256973, ...
```

```
In [18]: # We will drop ['qid1','qid2','question1','question2','is_duplicate'] from df:
         df3 = df.drop(['qid1','qid2','question1','question2','is_duplicate'], axis=1)
         df3.head()
```

```
Out[18]:    id                                         q1_feats_m  \
         0   0  [122.490798712, 100.359120488, 72.0331508666, ...
         1   1  [-74.5846772194, 53.8620963991, 81.0885115862,...
         2   2  [-5.10626339912, 73.7096084356, 14.3268437684,...
         3   3  [5.90131050348, -34.4693912566, 48.9884575009,...
         4   4  [48.4207775295, 38.2941785157, 121.9611063, 54...

                                                   q2_feats_m
         0  [126.564217329, 96.0618406534, 42.2021160275, ...
         1  [-105.099983424, 79.1588504314, 77.5340879094,...
         2  [6.49532223493, 16.2452982366, 2.65493392944, ...
         3  [38.9078674316, 43.9539289773, -24.3469197154,...
         4  [31.6172962189, 62.5719087124, 1.96994256973, ...
```

```
In [20]: # q1_feats_m has each row as a list. Therefore, we will extract it into a
         # dataframe as:
         st = time()
         df_q1 = pd.DataFrame(df3.q1_feats_m.values.tolist(), index = df3.index)
         time_taken(st)
         df_q1.head()
```

```
~> Time taken: 14.04 seconds
```

```
Out[20]:           0           1           2           3          4          5    \
         0  122.490799  100.359120   72.033151  115.891096 -48.144981  34.736722
         1  -74.584677   53.862096   81.088512   98.550397 -50.356915  53.286622
         2   -5.106263   73.709608   14.326844  104.493053   1.258413  35.409146
         3    5.901311  -34.469391   48.988458   59.481399  40.695803 -41.397960
         4   48.420778   38.294179  121.961106   54.678226 -45.466374  38.553049

                   6           7           8          9      ...           374  \
         0 -172.386330  -93.059744  113.417203  51.259765     ...     12.462868
         1  -37.665547  -82.297257   45.744834  -8.385913     ...    -21.548015
         2 -149.265339  -97.636930   42.259155  51.435161     ...      3.012211
         3  -36.726121   24.031034    0.295455 -29.501785     ...     13.059348
         4 -294.462586 -105.776589  103.886341  65.766421     ...     13.320748

                  375        376        377        378        379        380       381  \
         0  41.063396   8.037371 -15.198150  18.056487   6.217941 -30.221076  3.659344
         1 -11.906959  20.344241   1.829228 -16.460159  -5.656435 -10.035233 -4.768943
         2  14.140741  -2.977540  -3.214739   4.373585   2.911802 -20.323167  9.798284
         3   1.411459  -1.874297  -7.867466  17.947856  12.057635 -10.482685  5.230752
         4  42.630676  11.245030 -21.892262  43.775802   8.189654 -34.812249  8.047953
```

```
                  382       383
         0  -1.687294 -1.825006
         1 -12.692666 -5.208524
         2  11.907082 -8.814535
         3  10.150245  5.845988
         4   9.497889  5.378521

         [5 rows x 384 columns]
```

```
In [21]: # q12_feats_m has each row as a list. Therefore, we will extract it into a
         # dataframe as:
         st = time()
         df_q2 = pd.DataFrame(df3.q2_feats_m.values.tolist(), index = df3.index)
         time_taken(st)
         df_q2.head()
```

```
~> Time taken: 14.85 seconds
```

```
Out[21]:          0          1          2          3          4           5  \
         0  126.564217  96.061841  42.202116  95.969994 -37.314082   39.737327
         1 -105.099983  79.158850  77.534088  58.330385 -41.438078  115.591662
         2    6.495322  16.245298   2.654934  86.827784 -34.626589   95.729673
         3   38.907867  43.953929 -24.346920  86.120009   0.079079   -9.801455
         4   31.617296  62.571909   1.969943  36.472732 -45.163165   66.659808

                     6           7           8          9    ...           374  \
         0 -148.516119  -88.340872  110.552041  62.843040    ...     16.188503
         1 -142.872375 -125.501038   23.816001  25.313954    ...     -4.432317
         2 -123.613627 -115.022091   53.958783  61.496209    ...      8.264448
         3  -60.949873  -37.361491   49.504973 -22.386544    ...      3.488654
         4 -105.894651  -22.777562   59.957627  62.017545    ...     -2.440844

                  375        376        377        378        379        380  \
         0  33.233713   6.971700 -14.820828  15.534945   8.205955 -25.256606
         1  -4.367793  41.101273  -0.930737 -15.686246  -7.275999   2.756560
         2  -2.244750  11.084606 -16.741266  14.854023  15.726977  -1.298039
         3   3.906499  13.387563  -6.640244   6.378005   6.028185   2.511873
         4  11.887040   8.019029 -15.028031   8.280575   1.703147  -6.503707

                  381        382        383
         0   1.552828   1.651827   0.267462
         1  -7.351970   3.103773   0.440425
         2  14.340431  11.669012  10.423255
         3  -3.830347   5.421078   6.161891
         4  11.263387  11.556818   2.500520

         [5 rows x 384 columns]
```

```
In [23]: 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:", df_q1.shape[1])
         print("Number of features in question2 w2v dataframe:", df_q2.shape[1])
         print("Number of features in the final dataframe:",\
               df1.shape[1] + df2.shape[1] + df_q1.shape[1] + df_q2.shape[1])
```

```
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 the final dataframe: 797
```

```
In [25]: st = time()

         # The following code might take some time to execute, depending on the system
         # configuration.
         if not os.path.isfile('final_features_100k.csv'):

             # Attach 'id' attribute to question1 and question2 w2v vectors:
             df_q1['id'] = df1['id']
             df_q2['id'] = df1['id']

             # Merge nlp_features with preprocessing_features:
             df1 = df1.merge(df2, on='id', how='left')

             # Merge question1 and question2 w2v vectors and save them in df2 variable:
             df2 = df_q1.merge(df_q2, on='id', how='left')

             # We will now merge df1 and df2 into result:
             result = df1.merge(df2, on='id', how='left')

             # Save as a .csv file to use when applying k-NN to classify the points:
             result.to_csv('final_features_100k.csv')

         time_taken(st)
```

```
~> Time taken: 0.0 seconds
```

final_features_100k.csv file is generated, which is of 1.37GB. The file will be used to apply k-NN, so that we will be able to know how accurate the k-NN classifier is.