Training_Bayes_Classifier

November 30, 2019

0.1 Notebook Imports

```
In [1]: import pandas as pd
        import numpy as np
```

0.2 Constants

```
In [2]: TRAINING_DATA_FILE = 'SpamData/02_Training/train-data.txt'
        TEST_DATA_FILE = 'SpamData/02_Training/test-data.txt'

        TOKEN_SPAM_PROB_FILE = 'SpamData/03_Testing/prob-spam.txt'
        TOKEN_HAM_PROB_FILE = 'SpamData/03_Testing/prob-nonspam.txt'
        TOKEN_ALL_PROB_FILE = 'SpamData/03_Testing/prob-all-tokens.txt'

        TEST_FEATURE_MATRIX = 'SpamData/03_Testing/test-features.txt'
        TEST_TARGET_FILE = 'SpamData/03_Testing/test-target.txt'

        VOCAB_SIZE = 2500
```

0.3 Loading features from .txt file into NumPy Array

```
In [3]: sparse_train_data = np.loadtxt(TRAINING_DATA_FILE, delimiter=' ', dtype=int)

In [4]: sparse_test_data = np.loadtxt(TEST_DATA_FILE, delimiter=' ', dtype=int)

In [5]: sparse_train_data[:5]

Out[5]: array([[0, 0, 1, 1],
               [0, 1, 1, 1],
               [0, 2, 1, 1],
               [0, 6, 1, 1],
               [0, 9, 1, 1]])

In [6]: print('Number of rows in training file', sparse_train_data.shape[0])
        print('Number of rows in test file', sparse_test_data.shape[0])

Number of rows in training file 260245
Number of rows in test file 115650
```

```
In [7]: print('Number of emails in training file', np.unique(sparse_train_data[:, 0]).size)
```

```
Number of emails in training file 4017
```

```
In [8]: print('Number of emails in test file', np.unique(sparse_test_data[:, 0]).size)
```

```
Number of emails in test file 1722
```

0.4 Creating Full Matrix from Sparse Matrix

```
In [9]: def make_full_matrix(sparse_matrix, nr_words, doc_idx=0, word_idx=1, cat_idx=2, freq_i
            """
            Form a full matrix from a sparse matrix. Return a pandas dataframe.
            Keyword arguments:
            sparse_matrix -- numpy array
            nr_words -- size of the vocabulary. Total number of tokens.
            doc_idx -- position of the document id in the sparse matrix. Default: 1st column
            word_idx -- position of the word id in the sparse matrix. Default: 2nd column
            cat_idx -- position of the label (spam is 1, nonspam is 0). Default: 3rd column
            freq_idx -- position of occurrence of word in sparse matrix. Default: 4th column
            """
            column_names = ['DOC_ID'] + ['CATEGORY'] + list(range(0, VOCAB_SIZE))
            doc_id_names = np.unique(sparse_matrix[:, 0])
            full_matrix = pd.DataFrame(index=doc_id_names, columns=column_names)
            full_matrix.fillna(value=0, inplace=True)

            for i in range(sparse_matrix.shape[0]):
                doc_nr = sparse_matrix[i][doc_idx]
                word_id = sparse_matrix[i][word_idx]
                label = sparse_matrix[i][cat_idx]
                occurrence = sparse_matrix[i][freq_idx]

                full_matrix.at[doc_nr, 'DOC_ID'] = doc_nr
                full_matrix.at[doc_nr, 'CATEGORY'] = label
                full_matrix.at[doc_nr, word_id] = occurrence

            full_matrix.set_index('DOC_ID', inplace=True)
            return full_matrix
```

```
In [10]: %%time
         full_train_data = make_full_matrix(sparse_train_data, VOCAB_SIZE)
```

```
CPU times: user 8.82 s, sys: 177 ms, total: 9 s
Wall time: 8.75 s
```

```
In [11]: full_train_data.head()
```

```
Out[11]:         CATEGORY  0  1  2  3   4  5  6   7  8  ...   2490  2491  2492  2493  \
         DOC_ID                                         ...
         0              1  1  1  1  0   0  0  1   0  0  ...      0     0     0     0
         1              1  2  2  2  0   0  4  2   0  0  ...      0     0     0     0
         2              1  0  5  4  1  26  0  5  36  2  ...      0     0     0     0
         3              1  0  0  0  0   0  0  0   0  0  ...      0     0     0     0
         4              1  0  0  0  0   0  0  0   1  0  ...      0     0     0     0

                 2494  2495  2496  2497  2498  2499
         DOC_ID
         0          0     0     0     0     0     0
         1          0     0     0     0     0     0
         2          0     0     0     0     0     0
         3          0     0     0     0     0     0
         4          0     0     0     0     0     0

         [5 rows x 2501 columns]
```

1 Training the Naive Bayes Model

1.1 Calculating the probability of spam

```
In [12]: full_train_data.CATEGORY.size
```

```
Out[12]: 4017
```

```
In [13]: full_train_data.CATEGORY.sum()
```

```
Out[13]: 1252
```

```
In [14]: prob_spam = full_train_data.CATEGORY.sum()/full_train_data.CATEGORY.size
         print('Probability of spam is', prob_spam)
```

```
Probability of spam is 0.3116753796365447
```

1.2 Total Number of Words

```
In [15]: full_train_features = full_train_data.loc[:, full_train_data.columns != 'CATEGORY']
         full_train_features.head()
```

```
Out[15]:         0    1    2    3    4    5    6    7    8    9    ...  \
         DOC_ID                                                   ...
         0          1    1    1    0    0    0    1    0    0    1  ...
         1          2    2    2    0    0    4    2    0    0    2  ...
         2          0    5    4    1   26    0    5   36    2    0  ...
         3          0    0    0    0    0    0    0    0    0    0  ...
```

```
4          0     0     0     0     0     0     0     1     0     0  ...

        2490  2491  2492  2493  2494  2495  2496  2497  2498  2499
DOC_ID
0          0     0     0     0     0     0     0     0     0     0
1          0     0     0     0     0     0     0     0     0     0
2          0     0     0     0     0     0     0     0     0     0
3          0     0     0     0     0     0     0     0     0     0
4          0     0     0     0     0     0     0     0     0     0

[5 rows x 2500 columns]
```

```
In [16]: email_lengths = full_train_features.sum(axis=1)
         email_lengths.shape
```

```
Out[16]: (4017,)
```

```
In [17]: email_lengths[:5]
```

```
Out[17]: DOC_ID
         0     166
         1     165
         2    1367
         3      21
         4      24
         dtype: int64
```

```
In [18]: total_wc = email_lengths.sum()
         total_wc
```

```
Out[18]: 426844
```

1.3 Number of Tokens in Spam & Ham Emails

```
In [19]: spam_lengths = email_lengths[full_train_data.CATEGORY == 1]
         spam_lengths.shape
```

```
Out[19]: (1252,)
```

```
In [20]: spam_wc = spam_lengths.sum()
         spam_wc
```

```
Out[20]: 177155
```

```
In [21]: ham_lengths = email_lengths[full_train_data.CATEGORY == 0]
         ham_lengths.shape
```

```
Out[21]: (2765,)
```

```
In [22]: nonspam_wc = ham_lengths.sum()
         nonspam_wc
```

```
Out[22]: 249689

In [23]: spam_wc + nonspam_wc - total_wc

Out[23]: 0

In [24]: print('Average number of words in spam emails {:.0f}'.format(spam_wc / spam_lengths.sh
         print('Average number of words in ham emails {:.0f}'.format(nonspam_wc / ham_lengths.s

Average number of words in spam emails 141
Average number of words in ham emails 90
```

1.4 Summing the Tokens occuring in Spam

```
In [25]: full_train_features.shape

Out[25]: (4017, 2500)

In [26]: train_spam_tokens = full_train_features.loc[full_train_data.CATEGORY == 1]
         train_spam_tokens.head()

Out[26]:         0     1     2     3     4     5     6     7     8     9     ...   \
         DOC_ID                                                             ...
         0          1     1     1     0     0     0     1     0     0     1  ...
         1          2     2     2     0     0     4     2     0     0     2  ...
         2          0     5     4     1    26     0     5    36     2     0  ...
         3          0     0     0     0     0     0     0     0     0     0  ...
         4          0     0     0     0     0     0     0     1     0     0  ...

                 2490  2491  2492  2493  2494  2495  2496  2497  2498  2499
         DOC_ID
         0          0     0     0     0     0     0     0     0     0     0
         1          0     0     0     0     0     0     0     0     0     0
         2          0     0     0     0     0     0     0     0     0     0
         3          0     0     0     0     0     0     0     0     0     0
         4          0     0     0     0     0     0     0     0     0     0

         [5 rows x 2500 columns]

In [27]: train_spam_tokens.shape

Out[27]: (1252, 2500)

In [28]: summed_spam_tokens = train_spam_tokens.sum(axis=0) + 1
         summed_spam_tokens.shape

Out[28]: (2500,)

In [29]: summed_spam_tokens.tail()
```

```
Out[29]: 2495    24
         2496    23
         2497    18
         2498    11
         2499     9
         dtype: int64
```

1.5 Summing the Tokens Occuring in Ham

```
In [30]: train_ham_tokens = full_train_features.loc[full_train_data.CATEGORY == 0]
         summed_ham_tokens = train_ham_tokens.sum(axis=0) + 1
```

```
In [31]: summed_ham_tokens.shape
```

```
Out[31]: (2500,)
```

```
In [32]: summed_ham_tokens.tail()
```

```
Out[32]: 2495    11
         2496     1
         2497    20
         2498    12
         2499    18
         dtype: int64
```

1.6 P(word | Spam) - Probability that a Token Occurs given the Email is Spam

```
In [33]: prob_tokens_spam = summed_spam_tokens / (spam_wc + VOCAB_SIZE)
         prob_tokens_spam[:5]
```

```
Out[33]: 0    0.010303
         1    0.004859
         2    0.007832
         3    0.012262
         4    0.006835
         dtype: float64
```

1.7 P(word | Ham) - Probability that a Token Occurs given the Email is Non Spam

```
In [34]: prob_tokens_nonspam = summed_ham_tokens / (nonspam_wc + VOCAB_SIZE)
         prob_tokens_nonspam[:5]
```

```
Out[34]: 0    0.020877
         1    0.009937
         2    0.008069
         3    0.003549
         4    0.006626
         dtype: float64
```

1.8 P(Word) - Probability that Token Occurs

```
In [35]: prob_tokens_all = full_train_features.sum(axis=0) / total_wc
```

```
In [36]: prob_tokens_all.sum()
```

```
Out[36]: 1.0
```

1.9 Save the Trained Model

```
In [37]: np.savetxt(TOKEN_SPAM_PROB_FILE, prob_tokens_spam)
         np.savetxt(TOKEN_HAM_PROB_FILE, prob_tokens_nonspam)
         np.savetxt(TOKEN_ALL_PROB_FILE, prob_tokens_all)
```

1.10 Prepare Test Data

```
In [38]: sparse_test_data.shape
```

```
Out[38]: (115650, 4)
```

```
In [39]: %%time
         full_test_data = make_full_matrix(sparse_test_data, nr_words=VOCAB_SIZE)
```

```
CPU times: user 4.19 s, sys: 122 ms, total: 4.31 s
Wall time: 4.06 s
```

```
In [40]: X_test = full_test_data.loc[:, full_test_data.columns != 'CATEGORY']
         y_test = full_test_data.CATEGORY
```

```
In [41]: np.savetxt(TEST_TARGET_FILE, y_test)
         np.savetxt(TEST_FEATURE_MATRIX, X_test)
```