Assignment 3

May 11, 2021

---

*You are currently looking at **version 1.1** of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

1 Assignment 3

In this assignment you will explore text message data and create models to predict if a message is spam or not.

In [113]:
```
import pandas as pd
import numpy as np

spam_data = pd.read_csv('spam.csv')
spam_data['target'] = np.where(spam_data['target']=='spam',1,0)
spam_data.head(10)
```

Out[113]:

	text	target
0	Go until jurong point, crazy.. Available only ...	0
1	Ok lar... Joking wif u oni...	0
2	Free entry in 2 a wkly comp to win FA Cup fina...	1
3	U dun say so early hor... U c already then say...	0
4	Nah I don't think he goes to usf, he lives aro...	0
5	FreeMsg Hey there darling it's been 3 week's n...	1
6	Even my brother is not like to speak with me. ...	0
7	As per your request 'Melle Melle (Oru Minnamin...	0
8	WINNER!! As a valued network customer you have...	1
9	Had your mobile 11 months or more? U R entitle...	1

In [114]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(spam_data['text'],
                                                    spam_data['target'],
                                                    random_state=0)
```

1.0.1 Question 1

What percentage of the documents in `spam_data` are spam?

*This function should return a float, the percent value (i.e. ratio * 100).*

```
In [115]: def answer_one():

              spams = len(spam_data[spam_data['target'] == 1])
              return 100 * float(spams/len(spam_data))#Your answer here
```

```
In [116]: answer_one()
```

```
Out[116]: 13.406317300789663
```

1.0.2 Question 2

Fit the training data `X_train` using a Count Vectorizer with default parameters.

What is the longest token in the vocabulary?

This function should return a string.

```
In [117]: from sklearn.feature_extraction.text import CountVectorizer

          def answer_two():

              vect = CountVectorizer().fit(X_train)
              features = np.array(vect.get_feature_names())
              longest_string = max(features, key=len)
              return longest_string #Your answer here
```

```
In [118]: answer_two()
```

```
Out[118]: 'com1win150ppmx3age16subscription'
```

1.0.3 Question 3

Fit and transform the training data `X_train` using a Count Vectorizer with default parameters.

Next, fit a fit a multinomial Naive Bayes classifier model with smoothing `alpha=0.1`. Find the area under the curve (AUC) score using the transformed test data.

This function should return the AUC score as a float.

```
In [119]: from sklearn.naive_bayes import MultinomialNB
          from sklearn.metrics import roc_auc_score

          def answer_three():
              vect = CountVectorizer().fit(X_train)
              X_train_vectorized = vect.transform(X_train)
              X_test_vectorized = vect.transform(X_test)
              model = MultinomialNB(alpha=0.1)
              model.fit(X_train_vectorized,y_train)
              predictions = model.predict(X_test_vectorized)
              auc = roc_auc_score(y_test,predictions)
              return auc#Your answer here
```

```
In [120]: answer_three()
```

```
Out[120]: 0.97208121827411165
```

1.0.4 Question 4

Fit and transform the training data `X_train` using a Tfidf Vectorizer with default parameters.

What 20 features have the smallest tf-idf and what 20 have the largest tf-idf?

Put these features in a two series where each series is sorted by tf-idf value and then alphabetically by feature name. The index of the series should be the feature name, and the data should be the tf-idf.

The series of 20 features with smallest tf-idfs should be sorted smallest tfidf first, the list of 20 features with largest tf-idfs should be sorted largest first.

This function should return a tuple of two series (`smallest tf-idfs series, largest tf-idfs series`).

```
In [163]: from sklearn.feature_extraction.text import TfidfVectorizer

          def answer_four():
              import operator
              vect = TfidfVectorizer().fit(X_train)
              X_train_vectorized = vect.transform(X_train)
              features_names = vect.get_feature_names()
              idfs = vect.idf_
              names_idfs = list(zip(features_names, idfs))
              names_idfs = sorted(names_idfs, key=lambda x: x[1])
              smallest = sorted(names_idfs, key=operator.itemgetter(1))[:20]
              smallest = pd.Series([features[1] for features in smallest], index=[features[0] fo

              largest = sorted(names_idfs, key=operator.itemgetter(1), reverse=True)[:20]
              largest = sorted(largest, key=operator.itemgetter(0))
              largest = pd.Series([features[1] for features in largest], index=[features[0] for
              return (smallest, largest)
```

```
In [164]: answer_four()
```

```
Out[164]: (to      2.198406
           you     2.265645
           the     2.707383
           in      2.890761
           and     2.976764
           is      3.003012
           me      3.111530
           for     3.206840
           it      3.222174
           my      3.231044
           call    3.297812
           your    3.300196
           of      3.319473
```

```
have    3.354130
that    3.408477
on      3.463136
now     3.465949
can     3.545053
are     3.560414
so      3.566625
dtype: float64, 000pes         8.644919
0089           8.644919
0121           8.644919
01223585236    8.644919
0125698789     8.644919
02072069400    8.644919
02073162414    8.644919
02085076972    8.644919
021            8.644919
0430           8.644919
07008009200    8.644919
07099833605    8.644919
07123456789    8.644919
0721072        8.644919
07753741225    8.644919
077xxx         8.644919
078            8.644919
07808247860    8.644919
07808726822    8.644919
078498         8.644919
dtype: float64)
```

1.0.5 Question 5

Fit and transform the training data `X_train` using a Tfidf Vectorizer ignoring terms that have a document frequency strictly lower than **3**.

Then fit a multinomial Naive Bayes classifier model with smoothing `alpha=0.1` and compute the area under the curve (AUC) score using the transformed test data.

This function should return the AUC score as a float.

```
In [123]: def answer_five():

              vect = TfidfVectorizer(min_df = 3).fit(X_train)
              X_train_vectorized = vect.transform(X_train)
              X_test_vectorized = vect.transform(X_test)
              model = MultinomialNB(alpha=0.1)
              model.fit(X_train_vectorized,y_train)
              predictions = model.predict(X_test_vectorized)
              auc = roc_auc_score(y_test,predictions)
              return auc#Your answer here

In [124]: answer_five()
```

```
Out[124]: 0.94162436548223349
```

1.0.6 Question 6

What is the average length of documents (number of characters) for not spam and spam documents?

This function should return a tuple (average length not spam, average length spam).

```
In [125]: def answer_six():

              spam_data['char_count'] = spam_data['text'].apply(lambda x: len(x))
              df_len_not_spam = spam_data[spam_data['target'] == 0]
              average_len_not_spam = df_len_not_spam['char_count'].mean()
              df_len_spam = spam_data[spam_data['target'] == 1]
              average_len_spam = df_len_spam['char_count'].mean()
              return average_len_not_spam , average_len_spam #Your answer here
```

```
In [126]: answer_six()
```

```
Out[126]: (71.023626943005183, 138.8661311914324)
```

The following function has been provided to help you combine new features into the training data:

```
In [127]: def add_feature(X, feature_to_add):
              """
              Returns sparse feature matrix with added feature.
              feature_to_add can also be a list of features.
              """
              from scipy.sparse import csr_matrix, hstack
              return hstack([X, csr_matrix(feature_to_add).T], 'csr')
```

1.0.7 Question 7

Fit and transform the training data X_train using a Tfidf Vectorizer ignoring terms that have a document frequency strictly lower than **5**.

Using this document-term matrix and an additional feature, **the length of document (number of characters)**, fit a Support Vector Classification model with regularization `C=10000`. Then compute the area under the curve (AUC) score using the transformed test data.

This function should return the AUC score as a float.

```
In [128]: from sklearn.svm import SVC

          def answer_seven():
              vect = TfidfVectorizer(min_df = 5).fit(X_train)
              X_train_vectorized = vect.transform(X_train)
              X_train_vectorized_with_length = add_feature(X_train_vectorized, X_train.str.len()
              X_test_vectorized = vect.transform(X_test)
              X_test_vectorized_with_length = add_feature(X_test_vectorized, X_test.str.len())
```

```
    model = SVC(C=10000)
    model.fit(X_train_vectorized_with_length,y_train)
    predictions = model.predict(X_test_vectorized_with_length)
    auc = roc_auc_score(y_test,predictions)

    return auc#Your answer here
```

```
In [129]: answer_seven()
```

```
Out[129]: 0.95813668234215565
```

1.0.8 Question 8

What is the average number of digits per document for not spam and spam documents?

This function should return a tuple (average # digits not spam, average # digits spam).

```
In [130]: def answer_eight():

    spam_data['digit_count'] = spam_data['text'].apply(lambda x: len(''.join([a for a
    df_not_spam = spam_data[spam_data['target'] == 0]
    average_digit_not_spam = df_not_spam['digit_count'].mean()
    df_spam = spam_data[spam_data['target'] == 1]
    average_digit_spam = df_spam['digit_count'].mean()
    return average_digit_not_spam , average_digit_spam #Your answer here
```

```
In [131]: answer_eight()
```

```
Out[131]: (0.29927461139896372, 15.759036144578314)
```

1.0.9 Question 9

Fit and transform the training data `X_train` using a Tfidf Vectorizer ignoring terms that have a document frequency strictly lower than **5** and using **word n-grams from n=1 to n=3** (unigrams, bigrams, and trigrams).

Using this document-term matrix and the following additional features: * the length of document (number of characters) * **number of digits per document**

fit a Logistic Regression model with regularization `C=100`. Then compute the area under the curve (AUC) score using the transformed test data.

This function should return the AUC score as a float.

```
In [132]: from sklearn.linear_model import LogisticRegression

        def answer_nine():

            vect = TfidfVectorizer(min_df = 5,ngram_range=[1,3]).fit(X_train)
            X_train_vectorized = vect.transform(X_train)
            X_train_vectorized_with_length_digit = add_feature(X_train_vectorized, [X_train.st
            X_test_vectorized = vect.transform(X_test)
            X_test_vectorized_with_length_digit = add_feature(X_test_vectorized, [X_test.str.l
```

```
    model = LogisticRegression(C=100)
    model.fit(X_train_vectorized_with_length_digit,y_train)
    predictions = model.predict(X_test_vectorized_with_length_digit)
    auc = roc_auc_score(y_test,predictions)
    return auc#Your answer here
```

```
In [133]: answer_nine()
```

```
Out[133]: 0.96533283533945646
```

1.0.10 Question 10

What is the average number of non-word characters (anything other than a letter, digit or underscore) per document for not spam and spam documents?

Hint: Use \w and \W character classes

This function should return a tuple (average # non-word characters not spam, average # non-word characters spam).

```
In [141]: import re
          def answer_ten():
              spam_data['non_word_count'] = spam_data['text'].apply(lambda x: len(''.join([a for
              df_not_spam = spam_data[spam_data['target'] == 0]
              average_digit_not_spam = df_not_spam['non_word_count'].mean()
              df_spam = spam_data[spam_data['target'] == 1]
              average_digit_spam = df_spam['non_word_count'].mean()
              return average_digit_not_spam , average_digit_spam #Your answer here

              #return #Your answer here
```

```
In [142]: answer_ten()
```

```
Out[142]: (17.291813471502589, 29.041499330655956)
```

1.0.11 Question 11

Fit and transform the training data X_train using a Count Vectorizer ignoring terms that have a document frequency strictly lower than **5** and using **character n-grams from n=2 to n=5.**

To tell Count Vectorizer to use character n-grams pass in `analyzer='char_wb'` which creates character n-grams only from text inside word boundaries. This should make the model more robust to spelling mistakes.

Using this document-term matrix and the following additional features: * the length of document (number of characters) * number of digits per document * **number of non-word characters (anything other than a letter, digit or underscore.)**

fit a Logistic Regression model with regularization C=100. Then compute the area under the curve (AUC) score using the transformed test data.

Also **find the 10 smallest and 10 largest coefficients from the model** and return them along with the AUC score in a tuple.

The list of 10 smallest coefficients should be sorted smallest first, the list of 10 largest coefficients should be sorted largest first.

The three features that were added to the document term matrix should have the following names should they appear in the list of coefficients: ['length_of_doc', 'digit_count', 'non_word_char_count']

This function should return a tuple (AUC score as a float, smallest coefs list, largest coefs list).

In [153]:
```
def answer_eleven():

    vect = CountVectorizer(min_df = 5,ngram_range=[2,5], analyzer='char_wb').fit(X_tra
    X_train_vectorized = vect.transform(X_train)
    X_train_vectorized_with = add_feature(X_train_vectorized, [X_train.str.len(), X_tr
    X_test_vectorized = vect.transform(X_test)
    X_test_vectorized_with = add_feature(X_test_vectorized, [X_test.str.len(), X_test.
    model = LogisticRegression(C=100)
    model.fit(X_train_vectorized_with,y_train)
    predictions = model.predict(X_test_vectorized_with)
    auc = roc_auc_score(y_test,predictions)
    feature_names = np.array(vect.get_feature_names() + ['length_of_doc', 'digit_count
    sorted_coef_index = model.coef_[0].argsort()
    smallest = feature_names[sorted_coef_index[:10]]
    largest = feature_names[sorted_coef_index[:-11:-1]]


    return (auc, list(smallest), list(largest)[::-1])#Your answer here
```

In [154]:
```
answer_eleven()
```

Out[154]:
```
(0.97885931107074342,
 ['. ', '..', '? ', ' i', ' y', ' go', ':)', ' h', 'go', ' m'],
 ['ar', 'ww', ' x', 'mob', ' ch', 'xt', 'co', 'ia', 'ne', 'digit_count'])
```

In [ ]: