# Assignment 3

October 4, 2020

---

*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 [1]:
```
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[1]:

| | 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 [2]:
```
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 [5]: def answer_one():

            return spam_data['target'].mean()*100
```

```
In [6]: answer_one()
```

```
Out[6]: 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 [97]: from sklearn.feature_extraction.text import CountVectorizer

         def answer_two():

             vect = CountVectorizer().fit(X_train)

             feature_names = vect.get_feature_names()

             lengths = [len(token) for token in feature_names]

             token_lengths = sorted(zip(feature_names, lengths), key = lambda x: x[1], reverse =

             return token_lengths[0][0]
```

```
In [98]: answer_two()
```

```
Out[98]: '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 [99]: 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)

    classifier = MultinomialNB(alpha = 0.1).fit(X_train_vectorized, y_train)

    y_predict = classifier.predict_proba(vect.transform(X_test))[:,1]

    return roc_auc_score(y_test, y_predict)
```

```
In [70]: answer_three()
```

```
Out[70]: 0.99154542213469599
```

### 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 [161]: from sklearn.feature_extraction.text import TfidfVectorizer

          def answer_four():

              vect = TfidfVectorizer().fit(X_train)

              X_train_vectorized = vect.transform(X_train)

              feature_names = np.array(vect.get_feature_names())

              tfidf_values = X_train_vectorized.max(axis = 0).toarray()[0]

              sorted_tfidf_values = sorted(tfidf_values)

              sorted_tfidf_index = tfidf_values.argsort()

              smallest_tfidf = pd.Series(sorted_tfidf_values[:20], index = feature_names[sorted_

              largest_tfidf = pd.Series(sorted_tfidf_values[-21:-1], index = feature_names[sorte


              return (smallest_tfidf, largest_tfidf)
```

```
In [162]: answer_four()
```

```
Out[162]: (aaniye          0.074475
           athletic        0.074475
           chef            0.074475
           companion       0.074475
           courageous      0.074475
           dependable      0.074475
           determined      0.074475
           diwali          0.091250
           exterminator    0.074475
           healer          0.074475
           listener        0.074475
           mornings        0.091250
           organizer       0.074475
           pest            0.074475
           psychiatrist    0.074475
           psychologist    0.074475
           pudunga         0.074475
           stylist         0.074475
           sympathetic     0.074475
           venaam          0.074475
           dtype: float64, 645         1.000000
           anything    1.000000
           anytime     1.000000
           beerage     1.000000
           blank       0.932702
           done        1.000000
           er          1.000000
           havent      1.000000
           home        1.000000
           lei         1.000000
           nite        1.000000
           ok          1.000000
           okie        1.000000
           same        0.932467
           thank       1.000000
           thanx       1.000000
           tick        0.980166
           too         1.000000
           where       1.000000
           yup         1.000000
           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 [89]: def answer_five():

             vect = TfidfVectorizer(min_df = 3).fit(X_train)

             X_train_vectorized = vect.transform(X_train)

             clf = MultinomialNB(alpha= 0.1).fit(X_train_vectorized, y_train)

             y_predict = clf.predict_proba(vect.transform(X_test))[:,1]


             return roc_auc_score(y_test,y_predict)

In [90]: answer_five()

Out[90]: 0.99549683377756659
```

### 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 [173]: def answer_six():

              spam = spam_data['text'].where(spam_data['target']==1).dropna()

              not_spam = spam_data['text'].where(spam_data['target']==0).dropna()

              spam_length = [len(x) for x in spam]

              not_spam_length = [len(x) for x in not_spam]


              return (np.mean(not_spam_length), np.mean(spam_length))

In [174]: answer_six()

Out[174]: (71.023626943005183, 138.8661311914324)
```

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

```
In [219]: 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 [230]: from sklearn.svm import SVC

          def answer_seven():

              from sklearn.svm import SVC

              vect = TfidfVectorizer(min_df = 5).fit(X_train)

              X_train_vectorized = vect.transform(X_train)

              length_of_document_train = np.array([len(x) for x in X_train])

              clf = SVC(C=10000).fit(add_feature(X_train_vectorized,length_of_document_train), y

              X_test_vectorized = vect.transform(X_test)

              length_of_document_test = np.array([len(x) for x in X_test])

              y_predict = clf.decision_function(add_feature(X_test_vectorized,length_of_document

              return roc_auc_score(y_test,y_predict)

In [231]: answer_seven()

Out[231]: 0.99511060557187236
```

### 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 [278]: def answer_eight():

              import re

              spam = spam_data['text'].where(spam_data['target']==1).dropna().apply(lambda x: re

              not_spam = spam_data['text'].where(spam_data['target']==0).dropna().apply(lambda x

              spam_digits = [len(x) for x in spam]
```

```
    not_spam_digits = [len(x) for x in not_spam]

    return (np.mean(not_spam_digits), np.mean(spam_digits))
```

```
In [279]: answer_eight()
```

```
Out[279]: (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 [308]: from sklearn.linear_model import LogisticRegression

          def answer_nine():

              import re

              vect = TfidfVectorizer(min_df = 5, ngram_range = (1,3)).fit(X_train)

              X_train_vectorized = vect.transform(X_train)

              length_of_document_train = [len(x) for x in X_train]

              X_train_digits = X_train.apply(lambda x: re.sub('[^0-9]','', x)).apply(lambda x: x

              number_of_digits_train = [len(x) for x in X_train_digits]

              X_train_vect_aug = add_feature(add_feature(X_train_vectorized,length_of_document_t

              clf = LogisticRegression(C=100).fit(X_train_vect_aug, y_train)

              X_test_vectorized = vect.transform(X_test)

              length_of_document_test = [len(x) for x in X_test]

              X_test_digits = X_test.apply(lambda x: re.sub('[^0-9]','', x)).apply(lambda x: x.s

              number_of_digits_test = [len(x) for x in X_test_digits]

              X_test_vect_aug = add_feature(add_feature(X_test_vectorized,length_of_document_tes
```

```
    y_predict = clf.predict(X_test_vect_aug)

    return roc_auc_score(y_test,y_predict)
```

```
In [309]: answer_nine()
```

```
Out[309]: 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 [300]: def answer_ten():

    import re

    spam = spam_data['text'].where(spam_data['target']==1).dropna().apply(lambda x: re

    not_spam = spam_data['text'].where(spam_data['target']==0).dropna().apply(lambda x

    spam_non_words = [len(x) for x in spam]

    not_spam_non_words = [len(x) for x in not_spam]

    return (np.mean(not_spam_non_words), np.mean(spam_non_words))
```

```
In [301]: answer_ten()
```

```
Out[301]: (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 [325]: def answer_eleven():

              import re

              vect = CountVectorizer(min_df=5, ngram_range=(2,5), analyzer = 'char_wb').fit(X_tr

              X_train_vectorized = vect.transform(X_train)

              length_of_document_train = [len(x) for x in X_train]

              X_train_digits = X_train.apply(lambda x: re.sub('[^0-9]','', x)).apply(lambda x: x

              number_of_digits_train = [len(x) for x in X_train_digits]

              X_train_non_words = X_train.apply(lambda x: re.sub('[^\W]','', x))

              number_of_non_words_train = [len(x) for x in X_train_non_words]

              X_train_vect_aug = add_feature(add_feature(add_feature(X_train_vectorized,length_o

              clf = LogisticRegression(C=100).fit(X_train_vect_aug, y_train)



              X_test_vectorized = vect.transform(X_test)

              length_of_document_test = [len(x) for x in X_test]

              X_test_digits = X_test.apply(lambda x: re.sub('[^0-9]','', x)).apply(lambda x: x.s

              number_of_digits_test = [len(x) for x in X_test_digits]

              X_test_non_words = X_test.apply(lambda x: re.sub('[^\W]','', x))

              number_of_non_words_test = [len(x) for x in X_test_non_words]

              X_test_vect_aug = add_feature(add_feature(add_feature(X_test_vectorized,length_of_

              y_predict = clf.predict(X_test_vect_aug)
```

```
    roc_auc_score(y_test,y_predict)


    return (roc_auc_score(y_test,y_predict),sorted(clf.coef_[0])[:10],sorted(clf.coef_
```

In [326]:
```
answer_eleven()
```

Out[326]:
```
(0.97885931107074342,
 [-0.86975346259415875,
  -0.86088296352174276,
  -0.67698172297661741,
  -0.66700363899346971,
  -0.61490070391470086,
  -0.57958954863764711,
  -0.53507444581451002,
  -0.50576580840067331,
  -0.49850657095575379,
  -0.49096047527002135],
 [1.2122304448365826,
  0.59777832370761941,
  0.54148300240059377,
  0.53876611978762901,
  0.52148696634166225,
  0.52035142621300001,
  0.51786804549367671,
  0.51609606976459066,
  0.50867081207580411,
  0.50264408611612776])
```

In [ ]: