*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.*

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]:
```python
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]:
```python
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)
```

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 [3]:
```python
def answer_one():


    return len(spam_data[spam_data['target'] == 1]) / len(spam_data) * 100
```

In [4]:
```python
answer_one()
```

Out[4]:
```
13.406317300789663
```

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 [5]:
```python
from sklearn.feature_extraction.text import CountVectorizer

def answer_two():
    import operator

    vectorizer = CountVectorizer()
    vectorizer.fit(X_train)

    return sorted([(token, len(token)) for token in vectorizer.vocabulary_.keys()], key=operator.itemgetter(1), revers
e=True)[0][0]
```

In [6]:
```python
answer_two()
```

Out[6]:
```
'com1win150ppmx3age16subscription'
```

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 [7]:
```python
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

def answer_three():
    vectorizer = CountVectorizer()
    X_train_transformed = vectorizer.fit_transform(X_train)
    X_test_transformed = vectorizer.transform(X_test)

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

    y_predicted = clf.predict(X_test_transformed)

    return roc_auc_score(y_test, y_predicted)
```

In [8]:
```python
answer_three()
```

Out[8]:
```
0.97208121827411165
```

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 [9]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer

def answer_four():
    import operator

    vectorizer = TfidfVectorizer()
    X_train_transformed = vectorizer.fit_transform(X_train)

    feature_names = vectorizer.get_feature_names()
    idfs = vectorizer.idf_
    names_idfs = list(zip(feature_names, idfs))

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

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

    return (smallest, largest)
```

In [10]:
```python
answer_four()
```

Out[10]:
```
(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)
```

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 [11]:
```python
def answer_five():
    vectorizer = TfidfVectorizer(min_df=3)
    X_train_transformed = vectorizer.fit_transform(X_train)
    X_test_transformed = vectorizer.transform(X_test)

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

    # y_predicted_prob = clf.predict_proba(X_test_transformed)[:, 1]
    y_predicted = clf.predict(X_test_transformed)

    # return roc_auc_score(y_test, y_predicted_prob) #Your answer here
    return roc_auc_score(y_test, y_predicted)
```

In [12]:
```python
answer_five()
```

Out[12]:
```
0.94162436548223349
```

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 [13]:
```python
def answer_six():
    spam_data['length'] = spam_data['text'].apply(lambda x:len(x))

    return (np.mean(spam_data['length'][spam_data['target'] == 0]), np.mean(spam_data['length'][spam_data['target'] ==
1]))#Your answer here
```

In [14]:
```python
answer_six()
```

Out[14]:
```
(71.023626943005183, 138.8661311914324)
```

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

In [15]:
```python
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')
```

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 [16]:
```python
from sklearn.svm import SVC

def answer_seven():
    vectorizer = TfidfVectorizer(min_df=5)

    X_train_transformed = vectorizer.fit_transform(X_train)
    X_train_transformed_with_length = add_feature(X_train_transformed, X_train.str.len())

    X_test_transformed = vectorizer.transform(X_test)
    X_test_transformed_with_length = add_feature(X_test_transformed, X_test.str.len())

    clf = SVC(C=10000)

    clf.fit(X_train_transformed_with_length, y_train)

    y_predicted = clf.predict(X_test_transformed_with_length)

    return roc_auc_score(y_test, y_predicted)
```

In [17]:
```python
answer_seven()
```

Out[17]:
```
0.95813668234215565
```

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 [18]:
```python
def answer_eight():
    spam_data['length'] = spam_data['text'].apply(lambda x: len(''.join([a for a in x if a.isdigit()])))

    return (np.mean(spam_data['length'][spam_data['target'] == 0]), np.mean(spam_data['length'][spam_data['target'] ==
1]))
```

In [19]:
```python
answer_eight()
```

Out[19]:
```
(0.29927461139896372, 15.759036144578314)
```

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 [20]:
```python
from sklearn.linear_model import LogisticRegression

def answer_nine():
    vectorizer = TfidfVectorizer(min_df=5, ngram_range=[1,3])

    X_train_transformed = vectorizer.fit_transform(X_train)
    X_train_transformed_with_length = add_feature(X_train_transformed, [X_train.str.len(),
                                                                        X_train.apply(lambda x: len(''.join([a for a i
n x if a.isdigit()])))])

    X_test_transformed = vectorizer.transform(X_test)
    X_test_transformed_with_length = add_feature(X_test_transformed, [X_test.str.len(),
                                                                      X_test.apply(lambda x: len(''.join([a for a in x
if a.isdigit()])))])

    clf = LogisticRegression(C=100)

    clf.fit(X_train_transformed_with_length, y_train)

    y_predicted = clf.predict(X_test_transformed_with_length)

    return roc_auc_score(y_test, y_predicted)
```

In [21]:
```python
answer_nine()
```

Out[21]:
```
0.96533283533945646
```

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 [22]:
```python
def answer_ten():
    spam_data['length'] = spam_data['text'].str.findall(r'(\W)').str.len()

    return (np.mean(spam_data['length'][spam_data['target'] == 0]), np.mean(spam_data['length'][spam_data['target'] ==
1]))
```

In [23]:
```python
answer_ten()
```

Out[23]:
```
(17.291813471502589, 29.041499330655956)
```

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 [24]:
```python
def answer_eleven():
    vectorizer = CountVectorizer(min_df=5, analyzer='char_wb', ngram_range=[2,5])

    X_train_transformed = vectorizer.fit_transform(X_train)
    X_train_transformed_with_length = add_feature(X_train_transformed, [X_train.str.len(),
                                                                        X_train.apply(lambda x: len(''.join([a for a i
n x if a.isdigit()]))),
                                                                        X_train.str.findall(r'(\W)').str.len()])

    X_test_transformed = vectorizer.transform(X_test)
    X_test_transformed_with_length = add_feature(X_test_transformed, [X_test.str.len(),
                                                                      X_test.apply(lambda x: len(''.join([a for a in x
if a.isdigit()]))),
                                                                      X_test.str.findall(r'(\W)').str.len()])

    clf = LogisticRegression(C=100)

    clf.fit(X_train_transformed_with_length, y_train)

    y_predicted = clf.predict(X_test_transformed_with_length)

    auc = roc_auc_score(y_test, y_predicted)

    feature_names = np.array(vectorizer.get_feature_names() + ['length_of_doc', 'digit_count', 'non_word_char_count'])
    sorted_coef_index = clf.coef_[0].argsort()
    smallest = feature_names[sorted_coef_index[:10]]
    largest = feature_names[sorted_coef_index[:-11:-1]]

    return (auc, list(smallest), list(largest))
```

In [25]:
```python
answer_eleven()
```

Out[25]:
```
(0.97885931107074342,
 ['. ', '..', '? ', ' i', ' y', ' go', ':)', ' h', 'go', ' m'],
 ['digit_count', 'ne', 'ia', 'co', 'xt', ' ch', 'mob', ' x', 'ww', 'ar'])
```

In [ ]: