---

*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 (https://www.coursera.org/learn/python-text-mining/resources/d9pwm)](https://www.coursera.org/learn/python-text-mining/resources/d9pwm) 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 [28]:
```
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[28]:

	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 [29]:
```
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 [30]:
```
def answer_one():


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

In [31]:
```
answer_one()
```

Out[31]:
```
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 [32]:
```
from sklearn.feature_extraction.text import CountVectorizer

def answer_two():
    from sklearn.feature_extraction.text import CountVectorizer
    vect = CountVectorizer().fit(X_train)


    return max(vect.get_feature_names(), key=len)
```

In [33]:
```
answer_two()
```

Out[33]:
```
'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 [34]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

def answer_three():
    from sklearn.feature_extraction.text import CountVectorizer
    vect = CountVectorizer().fit(X_train)
    X_train_vectorized = vect.transform(X_train)

    model = MultinomialNB(alpha=0.1)
    model.fit(X_train_vectorized, y_train)
    y_pred = model.predict(vect.transform(X_test))

    return roc_auc_score(y_test, y_pred)
```

In [35]:
```
answer_three()
```

Out[35]: 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 [53]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

def answer_four():
    from sklearn.feature_extraction.text import TfidfVectorizer
    #fit vectorizer and create features from words
    vect = TfidfVectorizer().fit(X_train)
    #transform features into a sparse matrix with tfidf counts
    X_train_vectorized = vect.transform(X_train)
    #get feature names
    feature_names = np.array(vect.get_feature_names())
    #get max tfidf from each column and create array
    tfidf_index = X_train_vectorized.max(0).toarray()[0]
    #convert feature names and tfidf values to dataframe
    dfdata = {'words' : feature_names, 'tfidf' : tfidf_index}
    df_text = pd.DataFrame(dfdata)

    #sort df by tfidf
    df_text_sorted = df_text.sort_values('tfidf')
    smallest = df_text_sorted.head(20)
    largest = df_text_sorted.tail(20)

    #sort smallest values by tfidf and then by text, ascending order
    smallest = smallest.sort_values(['tfidf', 'words'])
    smallest = smallest.set_index('words')
    smallest_series = smallest['tfidf']
    smallest_series.index.name = None

    #sort largest values by tfidf and then by text, desceding order
    largest = largest.sort_values(['tfidf', 'words'], ascending=False)
    largest = largest.set_index('words')
    largest_series = largest['tfidf']
    largest_series.index.name = None

    return (smallest_series, largest_series)
```

In [54]:
```
answer_four()
```

Out[54]:
```
(aaniye          0.074475
 athletic        0.074475
 chef            0.074475
 companion       0.074475
 courageous      0.074475
 dependable      0.074475
 determined      0.074475
 exterminator    0.074475
 healer          0.074475
 listener        0.074475
 organizer       0.074475
 pest            0.074475
 psychiatrist    0.074475
 psychologist    0.074475
 pudunga         0.074475
 stylist         0.074475
 sympathetic     0.074475
 venaam          0.074475
 diwali          0.091250
 mornings        0.091250
 Name: tfidf, dtype: float64, yup          1.000000
 where        1.000000
 too          1.000000
 thanx        1.000000
 thank        1.000000
 okie         1.000000
 ok           1.000000
 nite         1.000000
 lei          1.000000
 home         1.000000
 havent       1.000000
 er           1.000000
 done         1.000000
 beerage      1.000000
 anytime      1.000000
 anything     1.000000
 645          1.000000
 146tf150p    1.000000
 tick         0.980166
 blank        0.932702
 Name: tfidf, 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 [38]:
```
def answer_five():
    from sklearn.feature_extraction.text import TfidfVectorizer
    #fit vectorizer and create features from words
    vect = TfidfVectorizer(min_df=3).fit(X_train)
    #transform features into a sparse matrix with tfidf counts
    X_train_vectorized = vect.transform(X_train)

    #Naive Bayes classifier
    from sklearn.naive_bayes import MultinomialNB
    model = MultinomialNB(alpha=0.1)
    model.fit(X_train_vectorized, y_train)
    y_pred = model.predict(vect.transform(X_test))


    return roc_auc_score(y_test, y_pred)
```

In [39]:
```
answer_five()
```

Out[39]: 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 [40]:
```
def answer_six():
    spam_data['length'] = spam_data['text'].str.len()
    not_spam = spam_data[spam_data['target'] == 0]
    spam = spam_data[spam_data['target'] == 1]

    return (not_spam['length'].mean(), spam['length'].mean())
```

In [41]:
```
answer_six()
```

Out[41]: (71.02362694300518, 138.8661311914324)

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

In [42]:
```
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 [43]:
```
from sklearn.svm import SVC

def answer_seven():
    spam_data['length_of_doc'] = spam_data['text'].str.len()
    spam_data['digit_count'] = spam_data['text'].str.findall(r'\d').str.len()
    spam_data['non_word_char_count'] = spam_data['text'].str.findall(r'\W').st
r.len()

    #Split
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(spam_data[['text', 'le
ngth_of_doc', 'digit_count', 'non_word_char_count']], spam_data['target'], ran
dom_state=0)

    from sklearn.feature_extraction.text import TfidfVectorizer
    #fit vectorizer and create features from words
    vect = TfidfVectorizer(min_df=5).fit(X_train['text'])
    #transform features into a sparse matrix with tfidf counts
    X_train_vectorized = vect.transform(X_train['text'])
    X_train_vectorized_added = add_feature(X_train_vectorized, X_train['length
_of_doc'])
    X_test_vectorized_added = add_feature(vect.transform(X_test['text']), X_te
st['length_of_doc'])

    #Use SVM
    from sklearn.svm import SVC
    from sklearn.metrics import roc_auc_score
    model = SVC(C=10000)
    model.fit(X_train_vectorized_added, y_train)
    y_pred = model.predict(X_test_vectorized_added)

    return roc_auc_score(y_test, y_pred)
```

In [44]:
```
answer_seven()
```

Out[44]: 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 [45]:
```
def answer_eight():

    spam_data['digits'] = spam_data['text'].str.findall(r'\d').str.len()
    not_spam = spam_data[spam_data['target'] == 0]
    not_spam['digits'].mean()

    spam = spam_data[spam_data['target'] == 1]
    spam['digits'].mean()

    return (not_spam['digits'].mean(), spam['digits'].mean() )
```

In [46]:
```
answer_eight()
```

Out[46]:
```
(0.2992746113989637, 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 [47]:
```
from sklearn.linear_model import LogisticRegression

def answer_nine():
    spam_data['length_of_doc'] = spam_data['text'].str.len()
    spam_data['digit_count'] = spam_data['text'].str.findall(r'\d').str.len()
    spam_data['non_word_char_count'] = spam_data['text'].str.findall(r'\W').st
r.len()

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(spam_data[['text', 'le
ngth_of_doc', 'digit_count', 'non_word_char_count']], spam_data['target'], ran
dom_state=0)

    from sklearn.feature_extraction.text import TfidfVectorizer
    #fit vectorizer and create features from words
    vect = TfidfVectorizer(min_df=5, ngram_range=(1,3)).fit(X_train['text'])
    #transform features into a sparse matrix with tfidf counts
    X_train_vectorized = vect.transform(X_train['text'])
    X_train_vectorized_added = add_feature(X_train_vectorized, [X_train['lengt
h_of_doc'],X_train['digit_count']])
    X_test_vectorized_added = add_feature(vect.transform(X_test['text']), [X_t
est['length_of_doc'],X_test['digit_count']])

    #Fit Logistic Regression
    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression(C=100)
    model.fit(X_train_vectorized_added, y_train)
    y_pred = model.predict(X_test_vectorized_added)

    return roc_auc_score(y_test, y_pred)
```

In [48]:
```
answer_nine()
```

Out[48]: 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 [49]:
```
def answer_ten():
    spam_data['non_word_char_count'] = spam_data['text'].str.findall(r'\W').st
r.len()
    not_spam = spam_data[spam_data['target'] == 0]
    spam = spam_data[spam_data['target'] == 1]

    return (not_spam['non_word_char_count'].mean(), spam['non_word_char_count'
].mean())
```

In [50]:
```
answer_ten()
```

Out[50]:
```
(17.29181347150259, 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 [51]:
```
def answer_eleven():
    import pandas as pd
    import numpy as np

    spam_data['length_of_doc'] = spam_data['text'].str.len()
    spam_data['digit_count'] = spam_data['text'].str.findall(r'\d').str.len()
    spam_data['non_word_char_count'] = spam_data['text'].str.findall(r'\W').st
r.len()

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(spam_data[['text', 'le
ngth_of_doc', 'digit_count', 'non_word_char_count']], spam_data['target'], ran
dom_state=0)

    from sklearn.feature_extraction.text import CountVectorizer
    #fit vectorizer and create features from words
    vect = CountVectorizer(min_df=5, ngram_range=(2,5), analyzer='char_wb').fi
t(X_train['text'])
    #transform features into a sparse matrix with tfidf counts
    X_train_vectorized = vect.transform(X_train['text'])
    X_train_vectorized_added = add_feature(X_train_vectorized, [X_train['lengt
h_of_doc'],X_train['digit_count'], X_train['non_word_char_count']])
    X_test_vectorized_added = add_feature(vect.transform(X_test['text']), [X_t
est['length_of_doc'],X_test['digit_count'], X_test['non_word_char_count']])

    #Fit Logistic Regression
    from sklearn.linear_model import LogisticRegression
    model = LogisticRegression(C=100)
    model.fit(X_train_vectorized_added, y_train)
    y_pred = model.predict(X_test_vectorized_added)

    roc_auc_score(y_pred, y_test)

    #Create lists
    features = vect.get_feature_names()
    coeffs = model.coef_
    #Since features does not contain the features added later, append them to
 list
    features.append('length_of_doc')
    features.append('digit_count')
    features.append('non_word_char_count')
    #Create dataframe and sort
    dfdata = {'features': features, 'coeffs': coeffs[0]}
    coeff_df = pd.DataFrame(dfdata)
    coeff_df = coeff_df.sort_values('coeffs')
    #create series of small coeffs
    small_coeff = coeff_df.head(10)
    small_coeff = small_coeff.set_index('features')
    small_coeff_series = pd.Series(small_coeff['coeffs'])

    #create series of large coeffs
    large_coeff = coeff_df.tail(10)
    large_coeff = large_coeff.sort_values('coeffs', ascending=False)
    large_coeff = large_coeff.set_index('features')
    large_coeff_series = pd.Series(large_coeff['coeffs'])
```

```
    return (roc_auc_score(y_test, y_pred),small_coeff_series, large_coeff_seri
es )
```

In [52]:
```
answer_eleven()
```

Out[52]:
```
(0.97885931107074342, features
 .     -0.869745
 ..    -0.860867
 ?     -0.676970
  i    -0.667010
  y    -0.614893
  go   -0.579597
 :)    -0.535072
  h    -0.505765
 go    -0.498514
  m    -0.490946
 Name: coeffs, dtype: float64, features
 digit_count    1.212221
 ne             0.597776
 ia             0.541486
 co             0.538758
 xt             0.521478
  ch            0.520357
 mob            0.517867
  x             0.516092
 ww             0.508665
 ar             0.502649
 Name: coeffs, dtype: float64)
```