Assignment3 (Score: 10.0 / 11.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 0.0 / 1.0)
4. Test cell (Score: 1.0 / 1.0)
5. Test cell (Score: 1.0 / 1.0)
6. Test cell (Score: 1.0 / 1.0)
7. Test cell (Score: 1.0 / 1.0)
8. Test cell (Score: 1.0 / 1.0)
9. Test cell (Score: 1.0 / 1.0)
10. Test cell (Score: 1.0 / 1.0)
11. Test cell (Score: 1.0 / 1.0)

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('assets/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)
```

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]:

Student's answer

```
def answer_one():
    return (len(spam_data[spam_data.target==1]) / len(spam_data)) * 100
answer_one()
```

Out[3]:

```
13.406317300789663
```

In [4]:

Grade cell: `cell-35ee2f1c33047f8c` Score: 1.0 / 1.0

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]:

Student's answer

```
from sklearn.feature_extraction.text import CountVectorizer
#import sklearn.feature_extraction.text.CountVectorizer
def answer_two():
    vect = CountVectorizer().fit(X_train)
    feature_names = np.array(vect.get_feature_names())
    length = list(map(len,feature_names))
    return feature_names[np.argmax(length)]
answer_two()
```

Out[5]:

```
'com1win150ppmx3age16subscription'
```

In [6]:

Grade cell: `cell-6eb97e449cd12bee` Score: 1.0 / 1.0

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 transformec data.

This function should return the AUC score as a float.

In [7]:

Student's answer

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import roc_auc_score

def answer_three():
    cv = CountVectorizer().fit(X_train)
    X_train_vect = cv.transform(X_train)
    X_test_vect = cv.transform(X_test)
    nbclf = MultinomialNB(alpha=0.1)
    nbclf.fit(X_train_vect, y_train)
    predicted_test = nbclf.predict(X_test_vect)

    return roc_auc_score(y_test, predicted_test)
answer_three()
```

Out[7]:

```
0.9720812182741116
```

In [8]:

Grade cell: `cell-c9dd3c3ec0b63773` Score: 0.0 / 1.0

```
You have failed this test due to an error. The traceback has been removed because it may contain hidden tests. This is

AssertionError: This is an incorrect solution
```

Question 4¶

Fit and transform the training data `X_train` using a Tfidf Vectorizer with default parameters. The transformed data will be a compressed sparse row r where the number of rows is the number of documents in `X_train` , the number of columns is the number of features found by the vectorizer in each document, and each value in the sparse matrix is the tf-idf value. First find the **max** tf-idf value for every feature.

What 20 features have the smallest tf-idf and what 20 have the largest tf-idf among the **max** tf-idf values?

Put these features in two series where each series is sorted by tf-idf value. The index of the series should be the feature name, and the data should be 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 entries with identical tf-ids should appear in lexigraphically increasing order by their feature name in boh series. For example, if the features "a", "b", "c the tf-idfs 1.0, 0.5, 1.0 in the series with the largest tf-idfs, then they should occur in the returned result in the order "a", "c", "b" with values 1.0, 1.0, 0.

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

In [9]:

Student's answer

```
from sklearn.feature_extraction.text import TfidfVectorizer

def answer_four():
    vect = TfidfVectorizer().fit(X_train)

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

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

    sorted_tfidf_index = X_train_vectorized.max(0).toarray()[0].argsort()

    small_index = feature_names[sorted_tfidf_index[:20]]
    small_value = np.sort(X_train_vectorized.max(0).toarray()[0])[:20]
    small_final_index = np.concatenate((np.sort(small_index[small_value==min(small_value)]) ,small_index[small_value!

    large_index = feature_names[sorted_tfidf_index[:-21:-1]]
    large_value = np.sort(X_train_vectorized.max(0).toarray()[0])[:-21:-1]
    large_final_index = np.concatenate((np.sort(large_index[large_value==max(large_value)]) ,large_index[large_value!

    small = pd.Series(small_value,index=small_final_index)
    large = pd.Series(large_value,index=large_final_index)
    return ((small,large))#Your answer here
answer_four()
```

Out[9]:

```
(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
 dtype: float64,
 146tf150p    1.000000
 645          1.000000
 anything     1.000000
 anytime      1.000000
 beerage      1.000000
 done         1.000000
 er           1.000000
 havent       1.000000
 home         1.000000
 lei          1.000000
 nite         1.000000
 ok           1.000000
 okie         1.000000
 thank        1.000000
 thanx        1.000000
 too          1.000000
 where        1.000000
 yup          1.000000
 tick         0.980166
 blank        0.932702
 dtype: float64)
```

In [10]:

Grade cell: `cell-09a122df96d70683` Score: 1.0 / 1.0

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 transfor test data.

This function should return the AUC score as a float.

In [11]:

Student's answer

```
def answer_five():
    vectorizer = TfidfVectorizer(min_df=3)
    X_train_vectorized = vectorizer.fit_transform(X_train)
    X_test_vectorized = vectorizer.transform(X_test)
    clf = MultinomialNB(alpha=0.1)
    clf.fit(X_train_vectorized, y_train)
    y_score = clf.predict_proba(X_test_vectorized)[:,1]



    return roc_auc_score(y_test, y_score)#Your answer here
answer_five()
```

Out[11]:

```
0.9954968337775665
```

In [12]:

Grade cell: `cell-09607be9c976cae0` Score: 1.0 / 1.0

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]:

Student's answer

```
def answer_six():
    length_spam = list(map(len,spam_data['text'][spam_data.target==1]))
    length_not_spam = list(map(len,spam_data['text'][spam_data.target==0]))

    return ((np.mean(length_not_spam),np.mean(length_spam)))#Your answer here
answer_six()
```

Out[13]:

```
(71.02362694300518, 138.8661311914324)
```

In [14]:

Grade cell: `cell-3cc7f12d3457b034` Score: 1.0 / 1.0

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

In [15]:

```
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 mode regularization `C=10000` . Then compute the area under the curve (AUC) score using the transformed test data.

Hint: Since probability is set to false, use the model's `decision_function` on the test data when calculating the target scores to use in roc_auc_scor

This function should return the AUC score as a float.

In [16]:

Student's answer

```
from sklearn.svm import SVC

def answer_seven():
    vectorizer = TfidfVectorizer(min_df=5)
    add_ftr_train = X_train.apply(lambda x: len(x))
    add_ftr_test = X_test.apply(lambda x: len(x))
    X_train_vectorized = vectorizer.fit_transform(X_train)
    X_test_vectorized = vectorizer.transform(X_test)
    X_train_added = add_feature(X_train_vectorized, add_ftr_train)
    X_test_added = add_feature(X_test_vectorized, add_ftr_test)

    clf = SVC(C=10000)
    clf.fit(X_train_added, y_train)
    y_score = clf.decision_function(X_test_added)

    return roc_auc_score(y_test, y_score)

answer_seven()
```

Out[16]:

```
0.9963202213809143
```

In [17]:

Grade cell: `cell-3627e3b7549e1a87` Score: 1.0 / 1.0

Question 8¶

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

Hint: Use `\d` for digit class

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

In [18]:

Student's answer

```
def answer_eight():
    import re
    spam = [re.findall("[0-9]",i) for i in spam_data['text'][spam_data.target==1]]
    non_spam = [re.findall("[0-9]",i) for i in spam_data['text'][spam_data.target==0]]

    return ((np.mean(list(map(len,non_spam))),np.mean(list(map(len,spam)))))#Your answer here
answer_eight()
```

Out[18]:

```
(0.2992746113989637, 15.759036144578314)
```

In [19]:

Grade cell: `cell-501bf7c435747a23` Score: 1.0 / 1.0

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 **wo 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` and `max_iter=1000`. Then compute the area under the curve (AUC) score using the transf test data.

This function should return the AUC score as a float.

In [20]:

Student's answer

```
from sklearn.linear_model import LogisticRegression

def answer_nine():
    vectorizer = TfidfVectorizer(min_df=5, ngram_range=(1,3))
    add_ftr_train1 = X_train.apply(lambda x: len(x))
    add_ftr_test1 = X_test.apply(lambda x: len(x))
    add_ftr_train2 = X_train.str.count(r'\d')
    add_ftr_test2 = X_test.str.count(r'\d')

    X_train_vectorized = vectorizer.fit_transform(X_train)
    X_test_vectorized = vectorizer.transform(X_test)

    X_train_added = add_feature(X_train_vectorized, add_ftr_train1)
    X_test_added = add_feature(X_test_vectorized, add_ftr_test1)

    X_train_added = add_feature(X_train_added, add_ftr_train2)
    X_test_added = add_feature(X_test_added, add_ftr_test2)

    clf = LogisticRegression(C=100, max_iter=1000)
    clf.fit(X_train_added, y_train)

    y_score = clf.predict_proba(X_test_added)[:, 1]

    return roc_auc_score(y_test, y_score)

answer_nine()
```

Out[20]:

```
0.9973218681561211
```

In [21]:

Grade cell: `cell-c7d3dd647af1574e` Score: 1.0 / 1.0

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 document

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]:

Student's answer

```
def answer_ten():
    import re
    spam = [re.findall("\W",i) for i in spam_data['text'][spam_data.target==1]]
    non_spam = [re.findall("\W",i) for i in spam_data['text'][spam_data.target==0]]

    return ((np.mean(list(map(len,non_spam))),np.mean(list(map(len,spam)))))#Your answer here
answer_ten()
```

Out[22]:

```
(17.29181347150259, 29.041499330655956)
```

In [23]:

Grade cell: `cell-1d6ac20393ffe9ff` Score: 1.0 / 1.0

Question 11¶

Fit and transform the **first 2000 rows** of training data X_train using a Count Vectorizer ignoring terms that have a document frequency strictly lower tha 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 boundai 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 and max_iter=1000. Then compute the area under the curve (AUC) score using the transform 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]:

Student's answer

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

    add_ftr_train1 = X_train[:2000].apply(lambda x: len(x))
    add_ftr_train2 = X_train[:2000].str.count(r'\d')
    add_ftr_train3 = X_train[:2000].str.count(r'\W')

    add_ftr_test1 = X_test.apply(lambda x: len(x))
    add_ftr_test2 = X_test.str.count(r'\d')
    add_ftr_test3 = X_test.str.count(r'\W')

    X_train_vectorized = vectorizer.fit_transform(X_train[:2000])
    X_test_vectorized = vectorizer.transform(X_test)

    X_train_added = add_feature(X_train_vectorized, add_ftr_train1)
    X_test_added = add_feature(X_test_vectorized, add_ftr_test1)

    X_train_added = add_feature(X_train_added, add_ftr_train2)
    X_test_added = add_feature(X_test_added, add_ftr_test2)

    X_train_added = add_feature(X_train_added, add_ftr_train3)
    X_test_added = add_feature(X_test_added, add_ftr_test3)

    clf = LogisticRegression(C=100, max_iter=1000)
    clf.fit(X_train_added, y_train[:2000])

    y_score = clf.predict_proba(X_test_added)[:, 1]
    auc = roc_auc_score(y_test, y_score)

    features = vectorizer.get_feature_names_out().tolist() + ['length_of_doc', 'digit_count', 'non_word_char_count']
    coefs = clf.coef_.tolist()[0]
    tupList = [(ftr, coef) for ftr, coef in zip(features, coefs)]
    tupList_sorted = sorted(tupList, key=lambda x: x[1])

    coef_smallest = [tup[0] for tup in tupList_sorted[:10]]
    coef_largest = [tup[0] for tup in tupList_sorted[-10:][::-1]]

    return auc, coef_smallest, coef_largest

answer_eleven()
```

Out[24]:

```
(0.997568035583926,
 ['n ', ' i', 'at', 'he', ' m', '..', 'us', 'go', ' lo', ' bu'],
 ['digit_count', 'ne', ' st', 'co', 's ', 'xt', 'lt', 'xt ', ' ne', 'der'])
```

In [ ]:

In [25]:

Grade cell: `cell-477ea85f5bcd7cef` Score: 1.0 / 1.0

This assignment was graded by mooc_adswpy:5a1483384bca, v1.48.110223