Assignment3 (Score: 11.0 / 11.0)

1. Test cell (Score: 1.0 / 1.0)
2. Test cell (Score: 1.0 / 1.0)
3. Test cell (Score: 1.0 / 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['tex
t'],
                                                    spam_data['targe
t'],
                                                    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 (Top)

```
def answer_one():

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

In [4]:
Grade cell: `cell-35ee2f1c33047f8c` Score: 1.0 / 1.0 (Top)

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 (Top)

```
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.vocab
ulary_.keys()], key=operator.itemgetter(1), reverse=True)[0][0]
```

In [6]:

Grade cell: `cell-6eb97e449cd12bee` Score: 1.0 / 1.0 (Top)

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]:

Student's answer (Top)

```
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]:

Grade cell: `cell-c9dd3c3ec0b63773` Score: 1.0 / 1.0 (Top)

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 matrix 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 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. Any 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" had 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, 0.5.

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

In [9]:

Student's answer (Top)

```
from sklearn.feature_extraction.text import TfidfVectorizer

def answer_four():

    tfidf_vect = TfidfVectorizer().fit(X_train)
    X_train_vectorized_tf = tfidf_vect.transform(X_train)

    feature_names_tf = np.array(tfidf_vect.get_feature_names())

    sorted_tfidf_index = X_train_vectorized_tf.max(0).toarray()[0].a
rgsort()
    sorted_tfidf_value = sorted(X_train_vectorized_tf.max(0).toarray
()[0])

    smallest = pd.Series(data=sorted_tfidf_value[:20], index=feature
_names_tf[sorted_tfidf_index[0:20]])

    largest = pd.Series(data=sorted_tfidf_value[:-21:-1], index=feat
ure_names_tf[sorted_tfidf_index[:-21:-1]])
    return (smallest,largest)
```

In [10]:

Grade cell: `cell-09a122df96d70683` Score: 1.0 / 1.0 (Top)

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]:

Student's answer (Top)

```
def answer_five():

    # YOUR CODE HERE
    vect = TfidfVectorizer(min_df=3).fit(X_train)
    X_train_vectorized = vect.transform(X_train)
    feature_names = np.array(vect.get_feature_names_out())

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

    predictions = model.predict_proba(vect.transform(X_test))[:,1]
    auc = roc_auc_score(y_test, predictions)
    # raise NotImplementedError()
    return auc #Your answer here

answer_five()
```

Out[11]: 0.9954968337775665

In [12]:

Grade cell: cell-09607be9c976cae0 Score: 1.0 / 1.0 (Top)

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 (Top)

```
def answer_six():

    #count text length ------------------------------------------
------------------
    spam_ave_len = spam_data[spam_data.target==1]['text'].apply(lamb
da x: len(x)).mean()
    not_spam_ave_len = spam_data[spam_data.target==0]['text'].apply
(lambda x: len(x)).mean()

    ans = (not_spam_ave_len,spam_ave_len)
    return(ans) #Your answer here
answer_six()
```

Out[13]: (71.02362694300518, 138.8661311914324)

In [14]:

Grade cell: cell-3cc7f12d3457b034 Score: 1.0 / 1.0 (Top)

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 model with 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_score

This function should return the AUC score as a float.

In [16]:

Student's answer (Top)

```
from sklearn.svm import SVC

def answer_seven():

    # YOUR CODE HERE
    vect= TfidfVectorizer(min_df=5).fit(X_train)
    X_train_vectorized = vect.transform(X_train)
    X_test_vectorized = vect.transform(X_test)

    doclength_train=X_train.apply(lambda x: len(x))
    X_train_vectorized=add_feature(X_train_vectorized,doclength_trai
n)

    doclength_test=X_test.apply(lambda x: len(x))
    X_test_vectorized=add_feature(X_test_vectorized,doclength_test)

    model=SVC(C=10000)
    # model.fit(X_train_vectorized,y_train)
    y_scores = model.fit(X_train_vectorized, y_train).decision_funct
ion(X_test_vectorized)
    auc = roc_auc_score(y_test, y_scores)

    # raise NotImplementedError()
    return auc #Your answer here

answer_seven()
```

Out[16]: 0.9963202213809143

In [17]:

Grade cell: cell-3627e3b7549e1a87 Score: 1.0 / 1.0 (Top)

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 (Top)

```
def answer_eight():

    ans = (spam_data[spam_data.target==0]['text'].str.count('\d').mean(),
           spam_data[spam_data.target==1]['text'].str.count('\d').mean())

    return(ans) #Your answer here
```

In [19]:

Grade cell: `cell-501bf7c435747a23` Score: 1.0 / 1.0 (Top)

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` and `max_iter=1000`. 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]:

Student's answer (Top)

```
from sklearn.linear_model import LogisticRegression

def answer_nine():

    # YOUR CODE HERE
    vect= TfidfVectorizer(min_df=5, ngram_range=(1,3)).fit(X_train)
    X_train_vectorized = vect.transform(X_train)
    X_test_vectorized = vect.transform(X_test)

    doclength_train=X_train.apply(lambda x: len(x))
    doclength_test=X_test.apply(lambda x: len(x))

    digetsperdoc_train=X_train.str.count('\d')
    digetsperdoc_test=X_test.str.count('\d')

    X_train_vectorized=add_feature(X_train_vectorized,[doclength_tra
in,digetsperdoc_train])
    X_test_vectorized=add_feature(X_test_vectorized,[doclength_test,
digetsperdoc_test])

    model=LogisticRegression(C=100,max_iter=1000).fit(X_train_vector
ized,y_train)
    predictions = model.predict_proba(X_test_vectorized)[:,1]
    auc = roc_auc_score(y_test, predictions)

    # raise NotImplementedError()
    return auc #Your answer here

answer_nine()
```

Out[20]: 0.9973006468261378

In [21]:

Grade cell: cell-c7d3dd647af1574e Score: 1.0 / 1.0 (Top)

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]:

Student's answer (Top)

```
def answer_ten():
    #count non character names using |W----------------------------
-------
    ans = (spam_data[spam_data.target==0]['text'].str.count('\W').me
an(),
           spam_data[spam_data.target==1]['text'].str.count('\W').me
an())

    return(ans) #Your answer here
```

In [23]:

Grade cell: `cell-1d6ac20393ffe9ff` Score: 1.0 / 1.0 (Top)

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 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 and max_iter=1000. 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]:

Student's answer (Top)

```
def answer_eleven():

    # YOUR CODE HERE
    vect= CountVectorizer(min_df=5, ngram_range=(2,5),analyzer='char
_wb').fit(X_train[:2000])
    X_train_vectorized = vect.transform(X_train[:2000])
    X_test_vectorized = vect.transform(X_test)

    doclength_train=X_train[:2000].apply(lambda x: len(x)).rename('l
ength_of_doc')
    doclength_test=X_test.apply(lambda x: len(x)).rename('length_of_
doc')

    digetsperdoc_train=X_train[:2000].str.count('\d').rename('digit_
count')
    digetsperdoc_test=X_test.str.count('\d').rename('digit_count')

    nwc_train=X_train[:2000].str.count('[^\w\d_]').rename('non_word_
char_count')
    nwc_test=X_test.str.count('[^\w\d_]').rename('non_word_char_coun
t')

    X_train_vectorized=add_feature(X_train_vectorized,[doclength_tra
in,digetsperdoc_train,nwc_train])
    X_test_vectorized=add_feature(X_test_vectorized,[doclength_test,
digetsperdoc_test,nwc_test])

    model=LogisticRegression(C=100,max_iter=1000).fit(X_train_vector
ized,y_train[:2000])
    predictions = model.predict_proba(X_test_vectorized)[:,1]
    auc = roc_auc_score(y_test, predictions)

    feature_names = np.append(np.array(vect.get_feature_names_out
()),['length_of_doc', 'digit_count', 'non_word_char_count'])
    sorted_coef_index = model.coef_[0].argsort()

    smallest=feature_names[sorted_coef_index[:10]].tolist()
    largest=feature_names[sorted_coef_index[:-11:-1]].tolist()

    # raise NotImplementedError()
    return (auc, smallest, largest) #Your answer here

answer_eleven()
```

Out[24]:

```
(0.9975637913179296,
 ['n ', ' i', 'at', 'he', ' m', '..', 'us', 'go', ' lo', ' bu'],
 ['digit_count', 'ne', ' st', 'co', 's ', 'xt', 'lt', 'xt ', ' ne', 'd
er'])
```

In [25]:

Grade cell: `cell-477ea85f5bcd7cef` Score: 1.0 / 1.0 (Top)

This assignment was graded by mooc_adswpy:c962dfa9e144, v1.33.013123