proj2

September 4, 2021

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook()
```

1 Project 2: Spam/Ham Classification

1.1 Feature Engineering, Logistic Regression, Cross Validation

1.2 Due Date: Monday 11/30, 11:59 PM PST

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the project, we ask that you **write your solutions individually**. If you do discuss the assignments with others please **include their names** at the top of your notebook.

Collaborators: *list collaborators here*

1.3 Disclaimer about `sns.distplot()`

This project was designed for a slightly older version of seaborn, which does not support the new `displot` method taught in Lecture 9. Instead, in this project will occasionally call `distplot` (with a `t`). As you may have noticed in several of the previous assignments, use of the `distplot` function triggers a deprecation warning to notify the user that they should replace all deprecated functions with the updated version. Generally, warnings should not be suppressed but we will do so in this assignment to avoid cluttering.

See the seaborn documentation on distributions and functions for more details.

[2]:
```
# Run this cell to suppress all FutureWarnings
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

1.4 Score Breakdown

Question	Points
1a	1
1b	1
1c	2

Question	Points
2	3
3a	2
3b	2
4	2
5	2
6a	1
6b	1
6c	2
6d	2
6e	1
6f	3
7	6
8	6
9	3
10	15
Total	55

2 Part I - Initial Analysis

[3]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
sns.set(style = "whitegrid",
        color_codes = True,
        font_scale = 1.5)
```

2.0.1 Loading in the Data

In email classification, our goal is to classify emails as spam or not spam (referred to as "ham") using features generated from the text in the email.

The dataset consists of email messages and their labels (0 for ham, 1 for spam). Your labeled training dataset contains 8348 labeled examples, and the unlabeled test set contains 1000 unlabeled examples.

Run the following cells to load in the data into DataFrames.

The `train` DataFrame contains labeled data that you will use to train your model. It contains four columns:

1. `id`: An identifier for the training example
2. `subject`: The subject of the email
3. `email`: The text of the email

4. `spam`: 1 if the email is spam, 0 if the email is ham (not spam)

The `test` DataFrame contains 1000 unlabeled emails. You will predict labels for these emails and submit your predictions to the autograder for evaluation.

[4]:
```
from utils import fetch_and_cache_gdrive
fetch_and_cache_gdrive('1SCASpLZFKCp2zek-toR3xeKX3DZnBSyp', 'train.csv')
fetch_and_cache_gdrive('1ZDFo9OTF96B5GP2Nzn8P8-AL7CTQXmC0', 'test.csv')

original_training_data = pd.read_csv('data/train.csv')
test = pd.read_csv('data/test.csv')

# Convert the emails to lower case as a first step to processing the text
original_training_data['email'] = original_training_data['email'].str.lower()
test['email'] = test['email'].str.lower()

original_training_data.head()
```

```
Using version already downloaded: Sat Nov 28 22:29:11 2020
MD5 hash of file: 0380c4cf72746622947b9ca5db9b8be8
Using version already downloaded: Sat Nov 28 22:29:12 2020
MD5 hash of file: a2e7abd8c7d9abf6e6fafc1d1f9ee6bf
```

[4]:

	id	subject	email	spam
0	0	Subject: A&L Daily to be auctioned in bankrupt…	url: http://boingboing.net/#85534171\n date: n…	0
1	1	Subject: Wired: "Stronger ties between ISPs an…	url: http://scriptingnews.userland.com/backiss…	0
2	2	Subject: It's just too small …	<html>\n <head>\n </head>\n <body>\n <font siz…	1
3	3	Subject: liberal defnitions\n	depends on how much over spending vs. how much…	0
4	4	Subject: RE: [ILUG] Newbie seeks advice - Suse…	hehe sorry but if you hit caps lock twice the …	0

2.0.2 Question 1a

First, let's check if our data contains any missing values. Fill in the cell below to print the number of NaN values in each column. If there are NaN values, replace them with appropriate filler values (i.e., NaN values in the `subject` or `email` columns should be replaced with empty strings). Print the number of NaN values in each column after this modification to verify that there are no NaN values left.

Note that while there are no NaN values in the `spam` column, we should be careful when replacing NaN labels. Doing so without consideration may introduce significant bias into our model when fitting.

The provided test checks that there are no missing values in your dataset.

[5]:
```
print(original_training_data.isnull().sum())
original_training_data = original_training_data.fillna('')
print(original_training_data.isnull().sum())
```

```
id         0
subject    6
email      0
spam       0
dtype: int64
id         0
subject    0
email      0
spam       0
dtype: int64
```

[6]:
```
grader.check("q1a")
```

[6]:

All tests passed!

2.0.3 Question 1b

In the cell below, print the text of the `email` field for the first ham and the first spam email in the original training set.

The provided tests just ensure that you have assigned `first_ham` and `first_spam` to rows in the data, but only the hidden tests check that you selected the correct observations.

[7]:
```
first_ham = original_training_data[original_training_data['spam'] ==␣
↪0]['email'].iloc[0]
first_spam = original_training_data[original_training_data['spam'] ==␣
↪1]['email'].iloc[0]
print(first_ham)
print(first_spam)
```

```
url: http://boingboing.net/#85534171
 date: not supplied

 arts and letters daily, a wonderful and dense blog, has folded up its tent due
 to the bankruptcy of its parent company. a&l daily will be auctioned off by the
 receivers. link[1] discuss[2] (_thanks, misha!_)

 [1] http://www.aldaily.com/
 [2] http://www.quicktopic.com/boing/h/zlfterjnd6jf
```

```
<html>
 <head>
 </head>
 <body>
 <font size=3d"4"><b> a man endowed with a 7-8" hammer is simply<br>
  better equipped than a man with a 5-6"hammer. <br>
 <br>would you rather have<br>more than enough to get the job done or fall =
 short. it's totally up<br>to you. our methods are guaranteed to increase y=
 our size by 1-3"<br> <a href=3d"http://209.163.187.47/cgi-bin/index.php?10=
 004">come in here and see how</a>
 </body>
 </html>
```

[8]:
```
grader.check("q1b")
```

[8]:
```
All tests passed!
```

2.0.4 Question 1c

Discuss one thing you notice that is different between the two emails that might relate to the identification of spam.

The spam emails have html tags that can be used to identify and separate the spam emails.

2.1 Training Validation Split

The training data we downloaded is all the data we have available for both training models and **validating** the models that we train. We therefore need to split the training data into separate training and validation datsets. You will need this **validation data** to assess the performance of your classifier once you are finished training. Note that we set the seed (random_state) to 42. This will produce a pseudo-random sequence of random numbers that is the same for every student. **Do not modify this in the following questions, as our tests depend on this random seed.**

[9]:
```
# This creates a 90/10 train-validation split on our labeled data

from sklearn.model_selection import train_test_split

train, val = train_test_split(original_training_data, test_size=0.1,␣
 ↪random_state=42)
```

3 Basic Feature Engineering

We would like to take the text of an email and predict whether the email is ham or spam. This is a *classification* problem, so we can use logistic regression to train a classifier. Recall that to train an logistic regression model we need a numeric feature matrix X and a vector of corresponding binary labels y. Unfortunately, our data are text, not numbers. To address this, we can create numeric features derived from the email text and use those features for logistic regression.

Each row of X is an email. Each column of X contains one feature for all the emails. We'll guide you through creating a simple feature, and you'll create more interesting ones as you try to increase the accuracy of your model.

3.0.1 Question 2

Create a function called `words_in_texts` that takes in a list of `words` and a pandas Series of email `texts`. It should output a 2-dimensional NumPy array containing one row for each email text. The row should contain either a 0 or a 1 for each word in the list: 0 if the word doesn't appear in the text and 1 if the word does. For example:

```
>>> words_in_texts(['hello', 'bye', 'world'],
                   pd.Series(['hello', 'hello worldhello']))

array([[1, 0, 0],
       [1, 0, 1]])
```

The provided tests make sure that your function works correctly, so that you can use it for future questions.

[10]:
```
def words_in_texts(words, texts):
    '''
    Args:
        words (list): words to find
        texts (Series): strings to search in

    Returns:
        NumPy array of 0s and 1s with shape (n, p) where n is the
        number of texts and p is the number of words.
    '''
    returnArray = []
    for text in texts:
        arr = []
        for word in words:
            arr.append(1 if word in text else 0)
        returnArray.append(arr)
    return returnArray
```

[11]:
```
grader.check("q2")
```

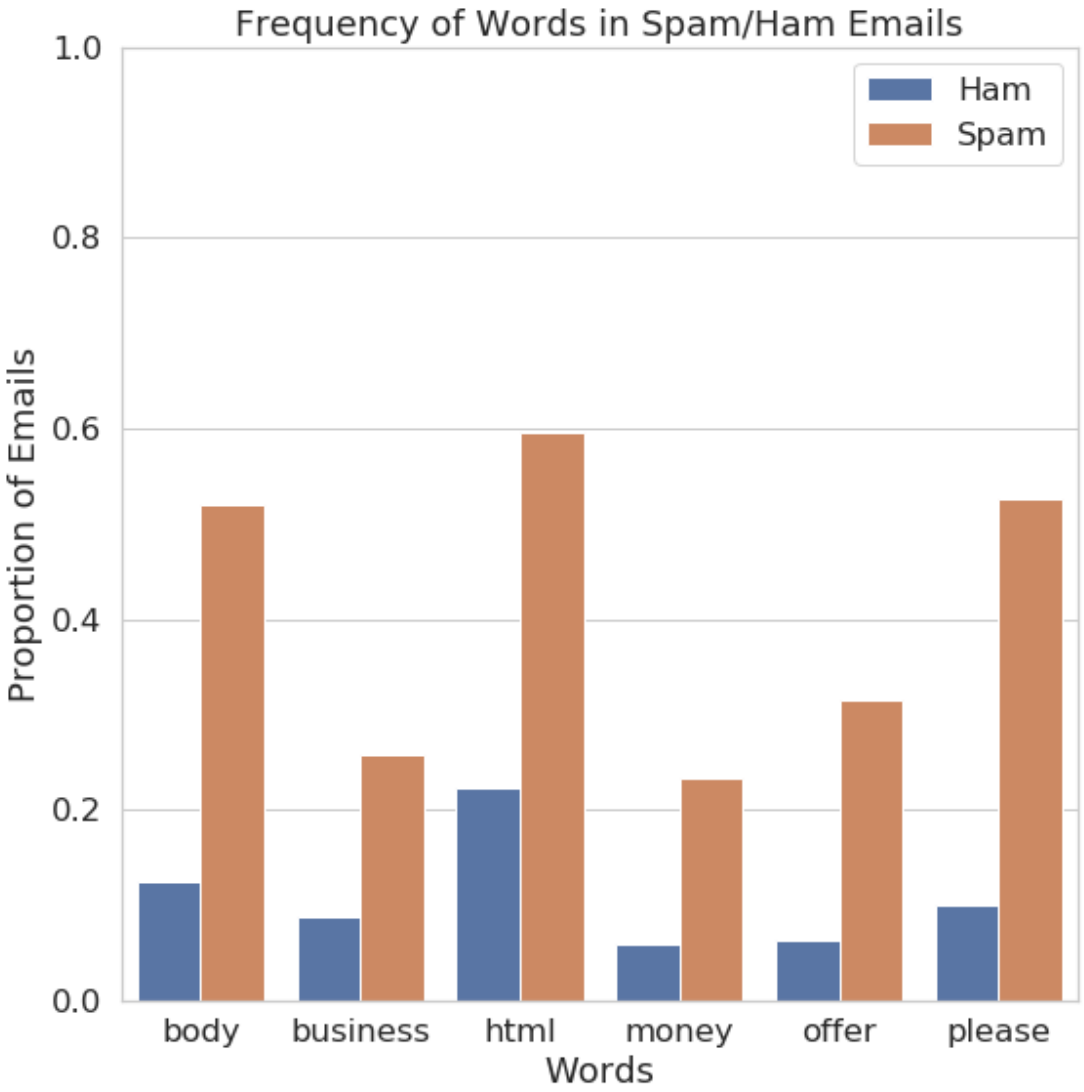
[11]:

```
All tests passed!
```

4 Basic EDA

We need to identify some features that allow us to distinguish spam emails from ham emails. One idea is to compare the distribution of a single feature in spam emails to the distribution of the same feature in ham emails. If the feature is itself a binary indicator, such as whether a certain word occurs in the text, this amounts to comparing the proportion of spam emails with the word to the proportion of ham emails with the word.

The following plot (which was created using `sns.barplot`) compares the proportion of emails in each class containing a particular set of words.



You can use DataFrame's `.melt` method to "unpivot" a DataFrame. See the following code cell for an example.

[12]:
```
from IPython.display import display, Markdown
df = pd.DataFrame({
    'word_1': [1, 0, 1, 0],
    'word_2': [0, 1, 0, 1],
    'type': ['spam', 'ham', 'ham', 'ham']
})
display(Markdown("> Our Original DataFrame has a `type` column and some columns␣
 ↪corresponding to words. You can think of each row as a sentence, and the␣
 ↪value of 1 or 0 indicates the number of occurences of the word in this␣
 ↪sentence."))
display(df);
display(Markdown("> `melt` will turn columns into entries in a variable column.␣
 ↪Notice how `word_1` and `word_2` become entries in `variable`; their values␣
 ↪are stored in the value column."))
display(df.melt("type"))
```

> Our Original DataFrame has a `type` column and some columns corresponding to words. You can think of each row as a sentence, and the value of 1 or 0 indicates the number of occurences of the word in this sentence.

	word_1	word_2	type
0	1	0	spam
1	0	1	ham
2	1	0	ham
3	0	1	ham

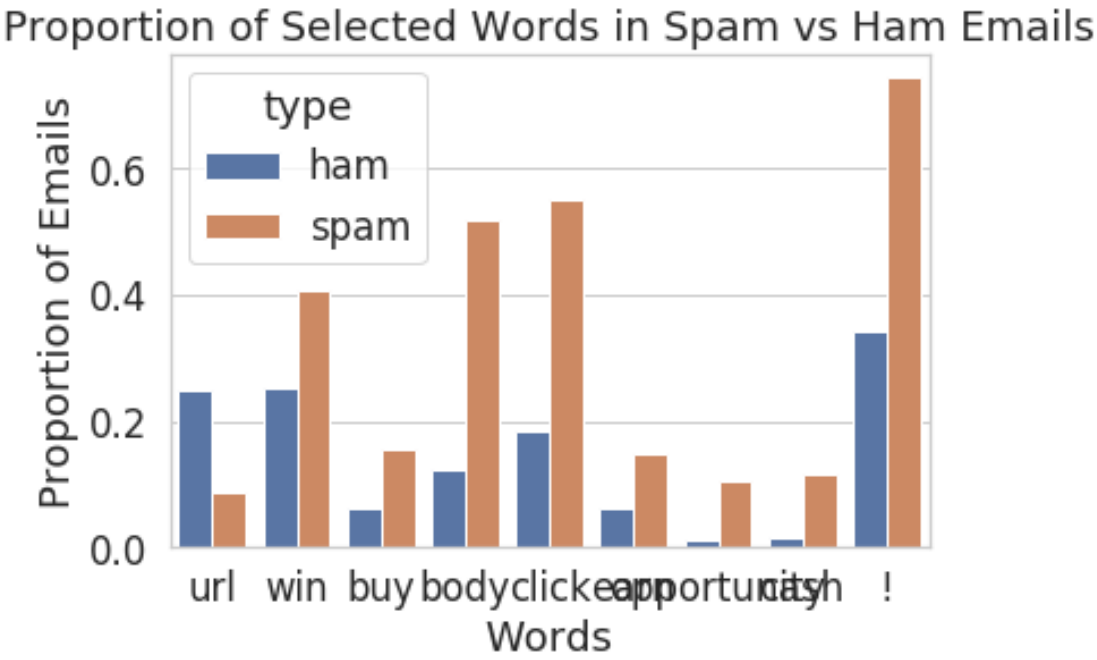
> `melt` will turn columns into entries in a variable column. Notice how `word_1` and `word_2` become entries in `variable`; their values are stored in the value column.

	type	variable	value
0	spam	word_1	1
1	ham	word_1	0
2	ham	word_1	1
3	ham	word_1	0
4	spam	word_2	0
5	ham	word_2	1
6	ham	word_2	0
7	ham	word_2	1

4.0.1 Question 3a

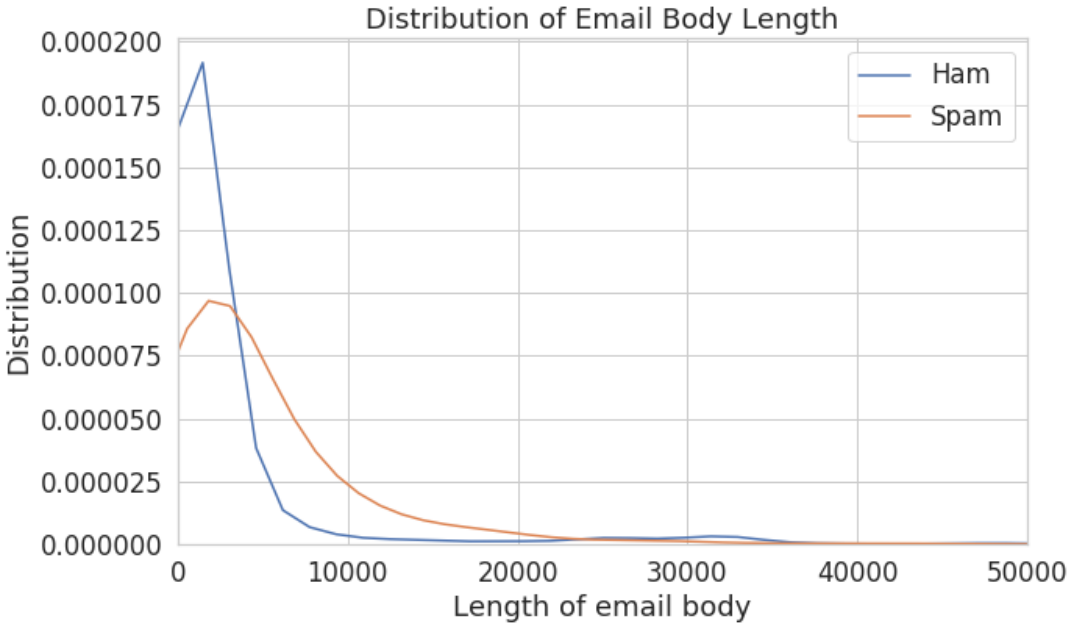
Create a bar chart like the one above comparing the proportion of spam and ham emails containing certain words. Choose a set of words that are different from the ones above, but also have different proportions for the two classes. Make sure to only consider emails from `train`.

[13]:
```
train=train.reset_index(drop=True) # We must do this in order to preserve the␣
↪ordering of emails to labels for words_in_texts

newWords = ['url', 'win', 'buy', 'body', 'click', 'earn', 'opportunity',␣
↪'cash', '!']
featureMatrix = words_in_texts(newWords, train['email'])
dataFrame = pd.DataFrame(data = featureMatrix, columns = newWords)
dataFrame['type'] = train['spam'].replace(0, 'ham').replace(1, 'spam')
dataFramePivoted = dataFrame.melt('type')
dataFramePivoted.groupby(['type','variable']).mean()
sns.barplot(x = 'variable', y = 'value', hue = 'type', data = dataFramePivoted,␣
↪ci = None)
plt.title('Proportion of Selected Words in Spam vs Ham Emails')
plt.ylabel('Proportion of Emails')
plt.xlabel('Words')
plt.show()
```



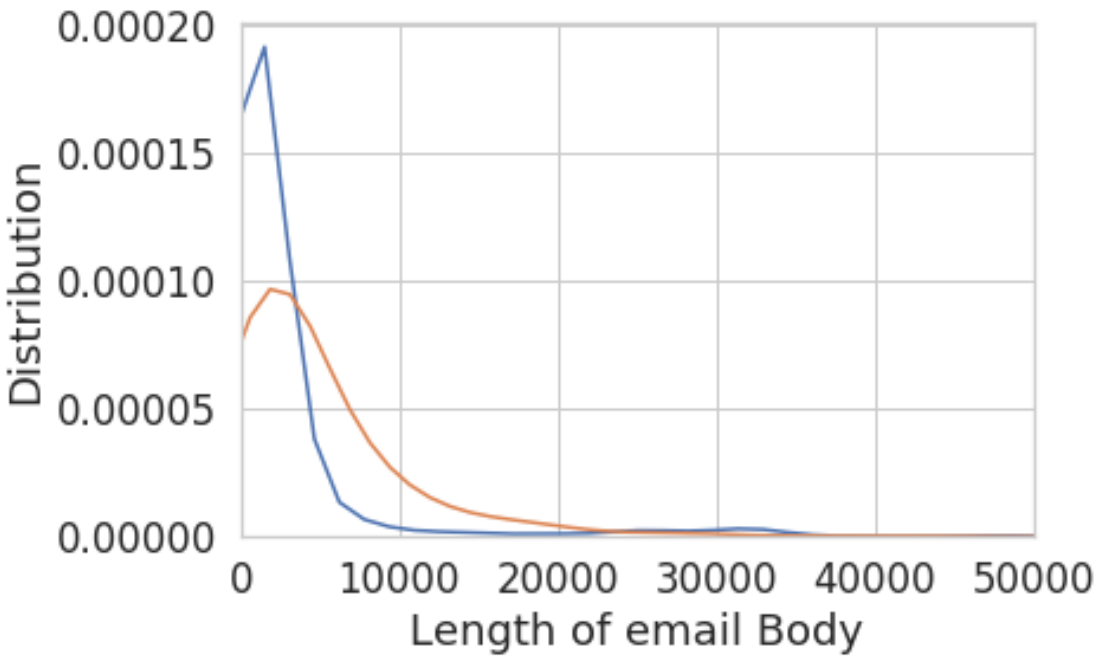
When the feature is binary, it makes sense to compare its proportions across classes (as in the previous question). Otherwise, if the feature can take on numeric values, we can compare the distributions of these values for different classes.

4.0.2 Question 3b



Create a *class conditional density plot* like the one above (using `sns.distplot`), comparing the distribution of the length of spam emails to the distribution of the length of ham emails in the training set. Set the x-axis limit from 0 to 50000.

[14]:
```
length = train['email'].apply(len)
tempDataFrame = pd.DataFrame({
    'spam': train['spam'],
    'length': length
})
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 0]['length'], hist = None,␣
↪label = 'Ham')
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 1]['length'], hist = None,␣
↪label = 'Spam')
plt.xlabel('Length of email Body')
plt.xlim([0,50000])
plt.ylabel('Distribution')
plt.savefig('training_conditional_densities.png')
```



5 Basic Classification

Notice that the output of `words_in_texts(words, train['email'])` is a numeric matrix containing features for each email. This means we can use it directly to train a classifier!

5.0.1 Question 4

We've given you 5 words that might be useful as features to distinguish spam/ham emails. Use these words as well as the `train` DataFrame to create two NumPy arrays: `X_train` and `Y_train`.

`X_train` should be a matrix of 0s and 1s created by using your `words_in_texts` function on all the emails in the training set.

`Y_train` should be a vector of the correct labels for each email in the training set.

The provided tests check that the dimensions of your feature matrix (X) are correct, and that your features and labels are binary (i.e. consists of only 0's and 1's). It does not check that your function is correct; that was verified in a previous question.

[15]:
```
some_words = ['drug', 'bank', 'prescription', 'memo', 'private']

X_train = np.array(words_in_texts(some_words, train['email']))
Y_train = np.array(train['spam'])

X_train[:5], Y_train[:5]
```

[15]:
```
(array([[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0]]),
 array([0, 0, 0, 0, 0]))
```

5.0.2 Question 5

Now that we have matrices, we can build a model with `scikit-learn`! Using the `LogisticRegression` classifier, train a logistic regression model using `X_train` and `Y_train`. Then, output the model's training accuracy below. You should get an accuracy of around 0.75

The provided test checks that you initialized your logistic regression model correctly.

[16]:
```
grader.check("q4")
```

[16]:
```
All tests passed!
```

[17]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(X_train, Y_train)

training_accuracy = model.score(X_train, Y_train)
print("Training Accuracy: ", training_accuracy)
```

```
Training Accuracy:  0.7576201251164648
```

[18]:
```
grader.check("q5")
```

[18]:
```
All tests passed!
```

5.1 Evaluating Classifiers

That doesn't seem too shabby! But the classifier you made above isn't as good as the accuracy would make you believe. First, we are evaluating accuracy on the training set, which may provide a misleading accuracy measure. Accuracy on the training set doesn't always translate to accuracy in the real world (on the test set). In future parts of this analysis, we will hold out some of our data for model validation and comparison.

Presumably, our classifier will be used for **filtering**, i.e. preventing messages labeled `spam` from reaching someone's inbox. There are two kinds of errors we can make: - False positive (FP): a ham email gets flagged as spam and filtered out of the inbox. - False negative (FN): a spam email gets mislabeled as ham and ends up in the inbox.

To be clear, we label spam emails as 1 and ham emails as 0. These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

Precision measures the proportion $\frac{\text{TP}}{\text{TP+FP}}$ of emails flagged as spam that are actually spam.

Recall measures the proportion $\frac{\text{TP}}{\text{TP+FN}}$ of spam emails that were correctly flagged as spam.

False-alarm rate measures the proportion $\frac{\text{FP}}{\text{FP+TN}}$ of ham emails that were incorrectly flagged as spam.

The two graphics below may help you understand precision and recall visually:

Note that a true positive (TP) is a spam email that is classified as spam, and a true negative (TN) is a ham email that is classified as ham.

5.1.1 Question 6a

Suppose we have a classifier `zero_predictor` that always predicts 0 (never predicts positive). How many false positives and false negatives would this classifier have if it were evaluated on the training set and its results were compared to `Y_train`? Fill in the variables below (feel free to hard code your answers for this part):

Tests in Question 6 only check that you have assigned appropriate types of values to each response variable, but do not check that your answers are correct.

[19]:
```
zero_predictor_fp = 0
zero_predictor_fn = np.count_nonzero(Y_train)
zero_predictor_fp, zero_predictor_fn
```

[19]:
```
(0, 1918)
```

[20]:
```
grader.check("q6a")
```

[20]:
```
All tests passed!
```

5.1.2 Question 6b

What is the accuracy and recall of `zero_predictor` (classifies every email as ham) on the training set? Do **NOT** use any `sklearn` functions.

[21]:
```
zero_predictor_acc = np.mean(Y_train == 0)
zero_predictor_recall = 0
zero_predictor_acc, zero_predictor_recall
```

[21]:
```
(0.7447091707706642, 0)
```

[22]:
```
grader.check("q6b")
```

[22]:
```
All tests passed!
```

5.1.3 Question 6c

Provide brief explanations of the results from 6a and 6b. Why do we observe each of these values (FP, FN, accuracy, recall)?

FP is 0 because false positives occur when a ham email is accidentally classified as spam and filtered out of the inbox. So when all emails are marked as ham, the zero_predictor only predicts 0, and the ham emails will never be marked as spam. Which means that the FP is 0.

The number of false negatives is 1918, which is also the amount of spam emails. False negatives occur when spam emails are marked as ham and appear in the inbox. Therefore, all spam emails are marked as ham since the zero_predictor only predicts ham.

Accuracy is 0.7447 and is the number of emails that are actually ham out of all the emails that were marked as ham. This is determined by the average of the number of zero values in Y_train since the zero_predictoe predicts all values are zero.

Recall is the number of relevant items, what we're looking for. So if the relevant items are spam emails, then none are chosen because the zero predictor classifier only predicts for ham emails.

5.1.4 Question 6d

Compute the precision, recall, and false-alarm rate of the `LogisticRegression` classifier created and trained in Question 5. Do **NOT** use any `sklearn` functions.

[23]:
```
Y_predict = model.predict(X_train)
FP = sum((Y_predict != Y_train) & (Y_predict == 1))
FN = sum((Y_predict != Y_train) & (Y_predict == 0))
TP = sum((Y_predict == Y_train) & (Y_predict == 1))
TN = sum((Y_predict == Y_train) & (Y_predict == 0))

print(FP)
print(FN)
logistic_predictor_precision = TP/(TP + FP)
logistic_predictor_recall = TP/(TP + FN)
logistic_predictor_far = FP/(FP + TN)
```

```
122
1699
```

[24]:
```
grader.check("q6d")
```

[24]:
```
All tests passed!
```

5.1.5 Question 6e

Are there more false positives or false negatives when using the logistic regression classifier from Question 5?

There are 1699 false negatives and 122 false positives. There are more false negatives than positives when using the logistic regression classifier.

5.1.6 Question 6f

1. Our logistic regression classifier got 75.76% prediction accuracy (number of correct predictions / total). How does this compare with predicting 0 for every email?
2. Given the word features we gave you above, name one reason this classifier is performing poorly. Hint: Think about how prevalent these words are in the email set.
3. Which of these two classifiers would you prefer for a spam filter and why? Describe your reasoning and relate it to at least one of the evaluation metrics you have computed so far.

The logistic regression classifier Acc at 75.6% has a higher prediction accuracy than the zero_prediction classifier at 74.5% which predicted that every single email was ham.

One reason the EDA classifier method is underperforming is because most of the words that it's trained on are only present in less less than 50% of the emails. For example, the words drug, bank, perscription, and private are much more likely to be classified as spam while only memo is more likely to be classified as ham.

The logistic regression classifier would be preferred because its recall is 0.114 vs. the zero predictor with a recall of 0. Therefore, the logistic regression classifier has a higher accuracy of correctly predicted spam emails.

6 Part II - Moving Forward

With this in mind, it is now your task to make the spam filter more accurate. In order to get full credit on the accuracy part of this assignment, you must get at least **88%** accuracy on the test set. To see your accuracy on the test set, you will use your classifier to predict every email in the `test` DataFrame and upload your predictions to Gradescope.

Gradescope limits you to four submissions per day. This means you should start early so you have time if needed to refine your model. You will be able to see your accuracy on 70% of the test set when submitting to Gradescope, but we will be evaluating your model on the entire test set so try to score slightly above 88% on gradescope if you can.

Here are some ideas for improving your model:

1. Finding better features based on the email text. Some example features are:
    1. Number of characters in the subject / body
    2. Number of words in the subject / body
    3. Use of punctuation (e.g., how many '!'s were there?)
    4. Number / percentage of capital letters
    5. Whether the email is a reply to an earlier email or a forwarded email
2. Finding better (and/or more) words to use as features. Which words are the best at distinguishing emails? This requires digging into the email text itself.

3. Better data processing. For example, many emails contain HTML as well as text. You can consider extracting out the text from the HTML to help you find better words. Or, you can match HTML tags themselves, or even some combination of the two.
4. Model selection. You can adjust parameters of your model (e.g. the regularization parameter) to achieve higher accuracy. Recall that you should use cross-validation to do feature and model selection properly! Otherwise, you will likely overfit to your training data.

You may use whatever method you prefer in order to create features, but **you are not allowed to import any external feature extraction libraries**. In addition, **you are only allowed to train logistic regression models**. No random forests, k-nearest-neighbors, neural nets, etc.

We have not provided any code to do this, so feel free to create as many cells as you need in order to tackle this task. However, answering questions 7, 8, and 9 should help guide you.

Note: *You may want to use your **validation data** to evaluate your model and get a better sense of how it will perform on the test set.* Note, however, that you may overfit to your validation set if you try to optimize your validation accuracy too much.

6.0.1 Question 7: Feature/Model Selection Process

In this following cell, describe the process of improving your model. You should use at least 2-3 sentences each to address the follow questions:

1. How did you find better features for your model?
2. What did you try that worked or didn't work?
3. What was surprising in your search for good features?

1) I used a variety of methods, firstly I logically thought through what typical spam email's goals are, which is almost certainly money related, so I tested those words in graphs we made earlier (Q3) and added them to the word bank. Then, question 8 helped me confirm my thoughts about spam emails not having proper html syntax and I also found some other apparent relationships from the heatmap, such a <p> and
 being grouped in ham emails.

2) Some words that I thought would be obvious features turned out not to be, as I originally tried the word Original, as I thought spam emails wouldn't generally have a Original Message but it turned out to not benefit as much as I thought.

3) The extra details I found in question 8 were really interesting, and were features that I did not think would exist.

6.0.2 Question 8: EDA

In the cell below, show a visualization that you used to select features for your model.

Include:

1. A plot showing something meaningful about the data that helped you during feature selection, model selection, or both.
2. Two or three sentences describing what you plotted and its implications with respect to your features.

Feel free to create as many plots as you want in your process of feature selection, but select only one for the response cell below.

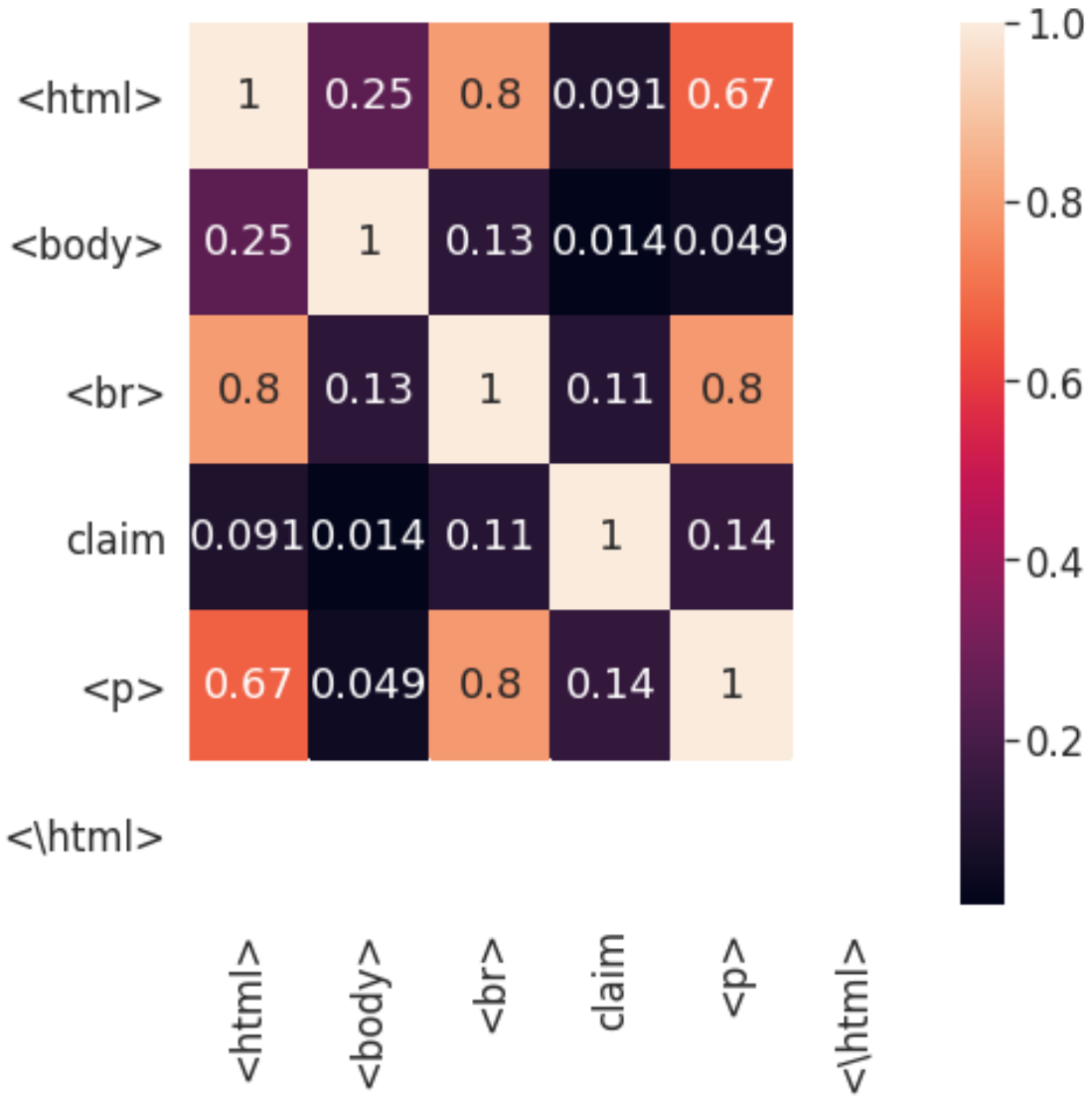
You should not just produce an identical visualization to question 3. Specifically, don't show us a bar chart of proportions, or a one-dimensional class-conditional density plot. Any other plot is acceptable, **as long as it comes with thoughtful commentary.** Here are some ideas:

1. Consider the correlation between multiple features (look up correlation plots and `sns.heatmap`).
2. Try to show redundancy in a group of features (e.g. `body` and `html` might co-occur relatively frequently, or you might be able to design a feature that captures all html tags and compare it to these).
3. Visualize which words have high or low values for some useful statistic.
4. Visually depict whether spam emails tend to be wordier (in some sense) than ham emails.

Generate your visualization in the cell below and provide your description in a comment.

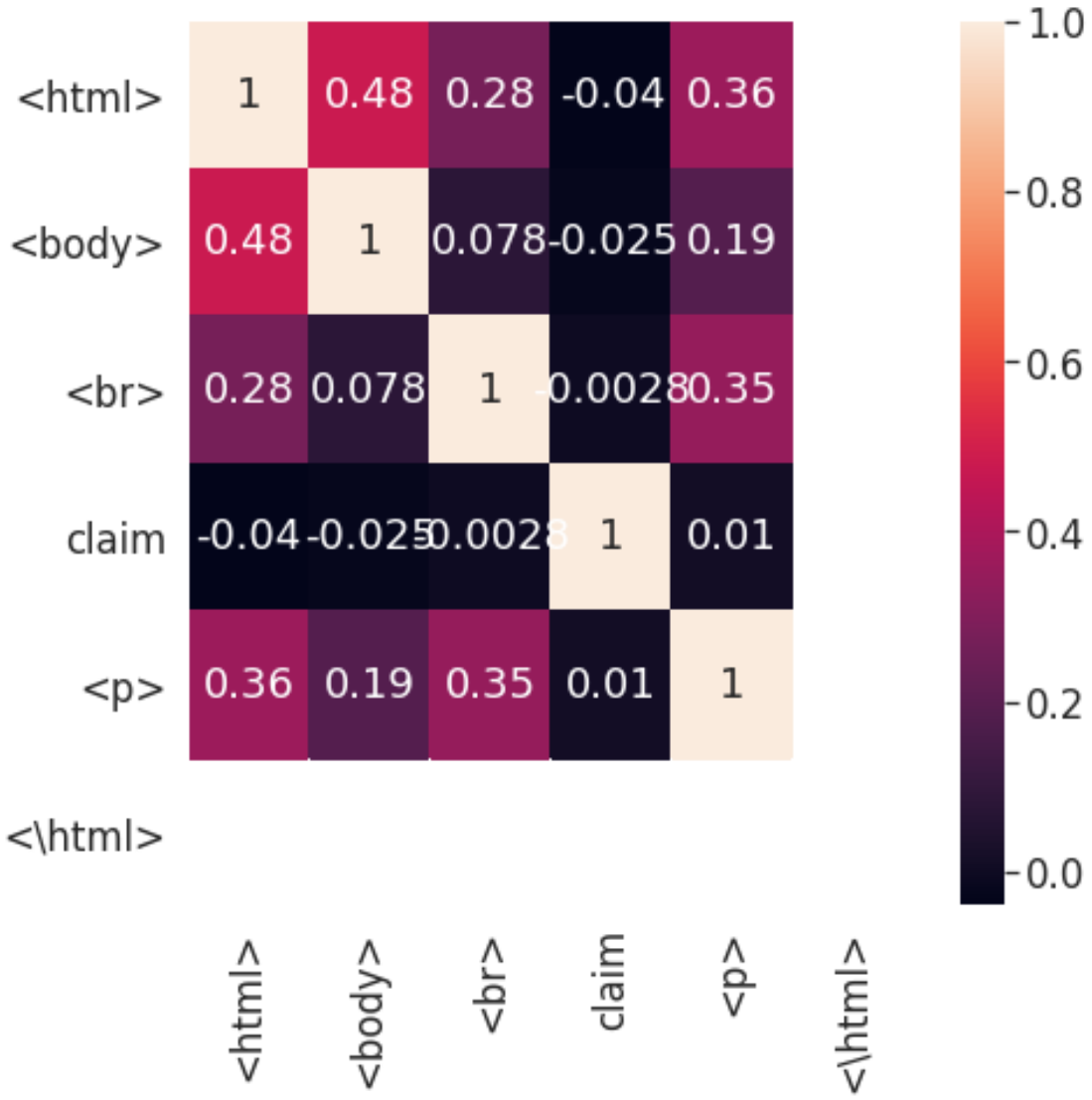
[25]:
```
# Write your description (2-3 sentences) as a comment here:
# I want to find out if different html tag combinations are used between ham␣
 ↪and spam emails.
# I made two heatmaps, one for the ham emails and one for the spam emails and␣
 ↪found that real emails often pair <html>
# tags and <br>, as well as <br> and <p> whereas spam emails don't have much␣
 ↪correlation between tags, meaning they may be more random.

def getWordCountArray(word, entrySet):
    returnArray = []
    for i in range(len(entrySet)):
        returnArray.append(entrySet[i].count(word))
    return np.array(returnArray)

tempDataFrame = pd.DataFrame({
    'spam': train['spam'],
    'email': train['email']
})
keyWords = ['<html>','<body>', '<br>', 'claim', '<p>', '<\html>']
matrix = []
for word in keyWords:
    matrix.append(getWordCountArray(word, np.array(train[train['spam'] ==␣
 ↪0]['email'])))

numpyMatrix = np.matrix(matrix)
pandasDataFrame = pd.DataFrame(data = numpyMatrix.T, columns = keyWords, )
corr = pandasDataFrame.corr()
plt.subplots(figsize=(7,7))
heatmap = sns.heatmap(corr, annot = True)

...
```

[25]: Ellipsis



[26]:
```
tempDataFrame = pd.DataFrame({
    'spam': train['spam'],
    'email': train['email']
})
keyWords = ['<html>','<body>', '<br>', 'claim', '<p>', '<\html>']
matrix = []
for word in keyWords:
    matrix.append(getWordCountArray(word, np.array(train[train['spam'] ==␣
 ↪1]['email'])))

numpyMatrix = np.matrix(matrix)
```

```
pandasDataFrame = pd.DataFrame(data = numpyMatrix.T, columns = keyWords, )
corr = pandasDataFrame.corr()
plt.subplots(figsize=(7,7))
heatmap = sns.heatmap(corr, annot = True)
```



6.0.3 Question 9: ROC Curve

In most cases we won't be able to get 0 false positives and 0 false negatives, so we have to compromise. For example, in the case of cancer screenings, false negatives are comparatively worse than false positives — a false negative means that a patient might not discover that they have cancer until it's too late, whereas a patient can just receive another screening for a false positive.

Recall that logistic regression calculates the probability that an example belongs to a certain class. Then, to classify an example we say that an email is spam if our classifier gives it ≥ 0.5 probability

of being spam. However, *we can adjust that cutoff*: we can say that an email is spam only if our classifier gives it ≥ 0.7 probability of being spam, for example. This is how we can trade off false positives and false negatives.

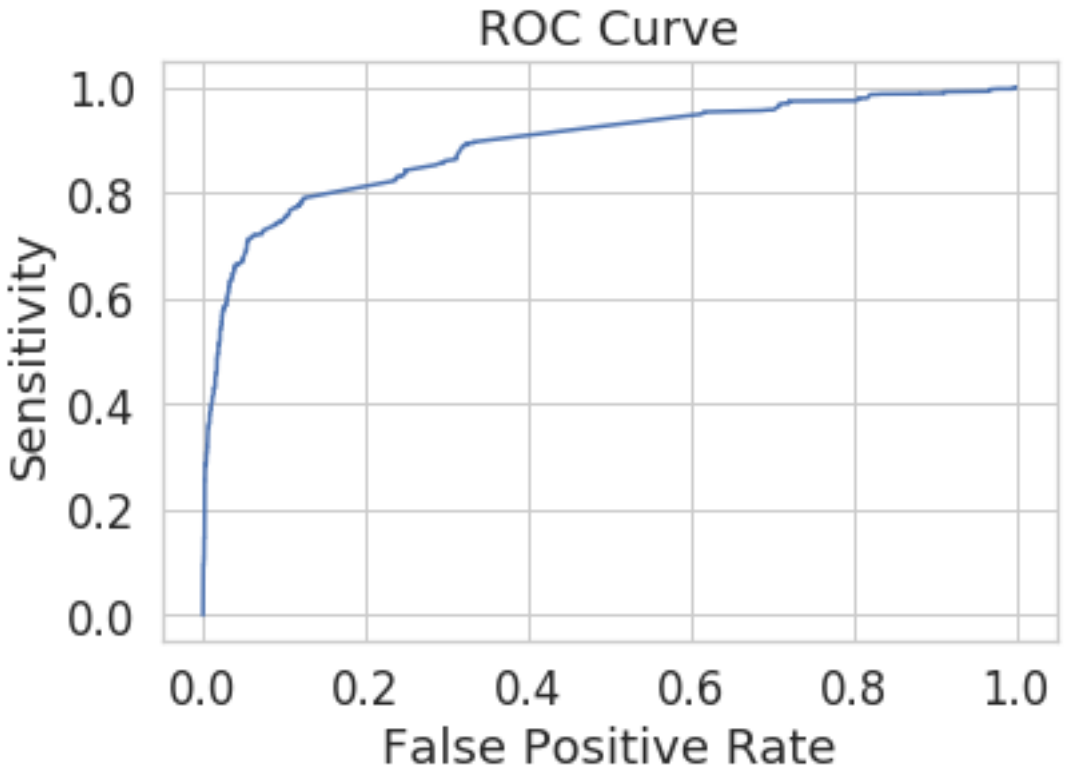
The ROC curve shows this trade off for each possible cutoff probability. In the cell below, plot a ROC curve for your final classifier (the one you use to make predictions for Gradescope) on the training data. Refer to Lecture 19 or Section 17.7 of the course text to see how to plot an ROC curve.

[27]:
```
#Helper function for results found in Q8
def getPairedHTMLTagsMatrix(texts):
    returnArray = []
    for text in texts:
        arr = []
        if '<html>' in text and '<\html>' in text:
            arr.append(1)
        else:
            arr.append(0)

        if '<html>' in text and '<br>' in text:
            arr.append(1)
        else:
            arr.append(0)

        if '<br>' in text and '<p>' in text:
            arr.append(1)
        else:
            arr.append(0)
        returnArray.append(arr)
    return returnArray
```

[28]:
```
from sklearn.metrics import roc_curve

# Note that you'll want to use the .predict_proba(...) method for your␣
 ↪classifier
# instead of .predict(...) so you get probabilities, not classes
model = LogisticRegression()
emailWords = ['<html>', 'money','url', 'business', 'click', 'subscribe',␣
 ↪'daily', 'win', 'buy', 'million', 'shipping', '!', 'earn', 'opportunity',␣
 ↪'cash', 'credit']
emailMatrix = words_in_texts(emailWords, train['email'])
subjectWords = ['RE:']
subjectMatrix = words_in_texts(subjectWords, train['subject'])
htmlSyntaxMatrix = getPairedHTMLTagsMatrix(train['email'])
finalMatrix = []

for i in range(len(emailMatrix)):
    finalMatrix.append(emailMatrix[i] + subjectMatrix[i] + htmlSyntaxMatrix[i])
```

```
xTrain = np.array(finalMatrix)
yTrain = np.array(train['spam'])
model.fit(xTrain, yTrain)

predictProbability = model.predict_proba(xTrain)[:, 1]
falsePositiveRate, sensitivity, thresholds = roc_curve(yTrain
, predictProbability)
plt.plot(falsePositiveRate, sensitivity)
plt.xlabel('False Positive Rate')
plt.ylabel('Sensitivity')
plt.title('ROC Curve')
...
```

[28]: Ellipsis



7 Question 10: Test Predictions

The following code will write your predictions on the test dataset to a CSV file. **You will need to submit this file to the "Project 2 Test Predictions" assignment on Gradescope to get credit for this question.**

Save your predictions in a 1-dimensional array called `test_predictions`. **Please make sure you've saved your predictions to `test_predictions` as this is how part of your score for this question will be determined.**

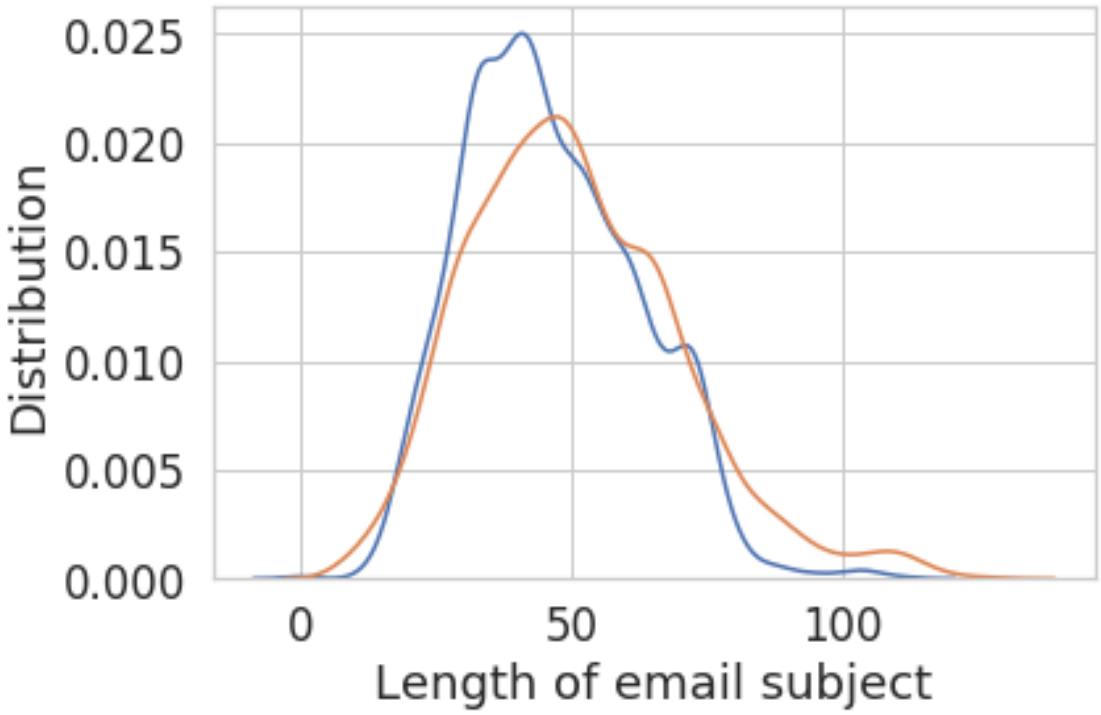
Remember that if you've performed transformations or featurization on the training data, you must also perform the same transformations on the test data in order to make predictions. For example, if you've created features for the words "drug" and "money" on the training data, you must also extract the same features in order to use scikit-learn's `.predict(...)` method.

Note: You may submit up to 4 times a day. If you have submitted 4 times on a day, you will need to wait until the next day for more submissions.

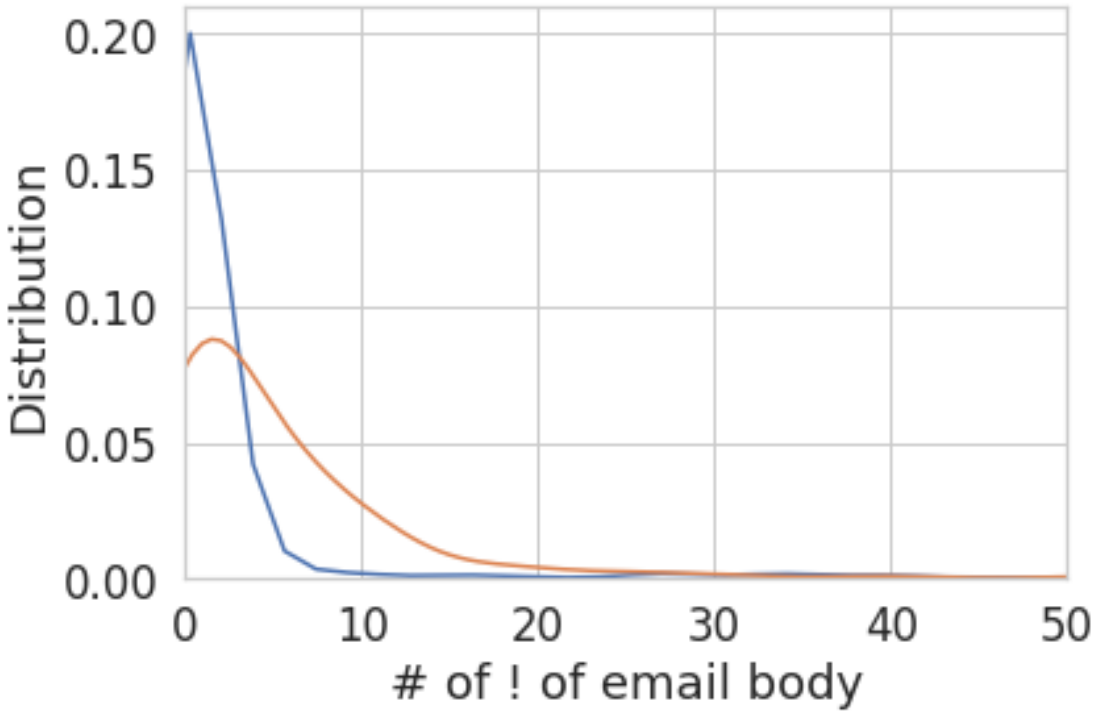
Note that this question is graded on an absolute scale based on the accuracy your model achieves on the overall test set, and as such, your score does not depend on your ranking on Gradescope. Your public Gradescope results are based off of your classifier's accuracy on 70% of the test dataset and your score for this question will be based off of your classifier's accuracy on 100% of the test set.

The provided tests check that your predictions are in the correct format, but you must additionally submit to Gradescope to evaluate your classifier accuracy.

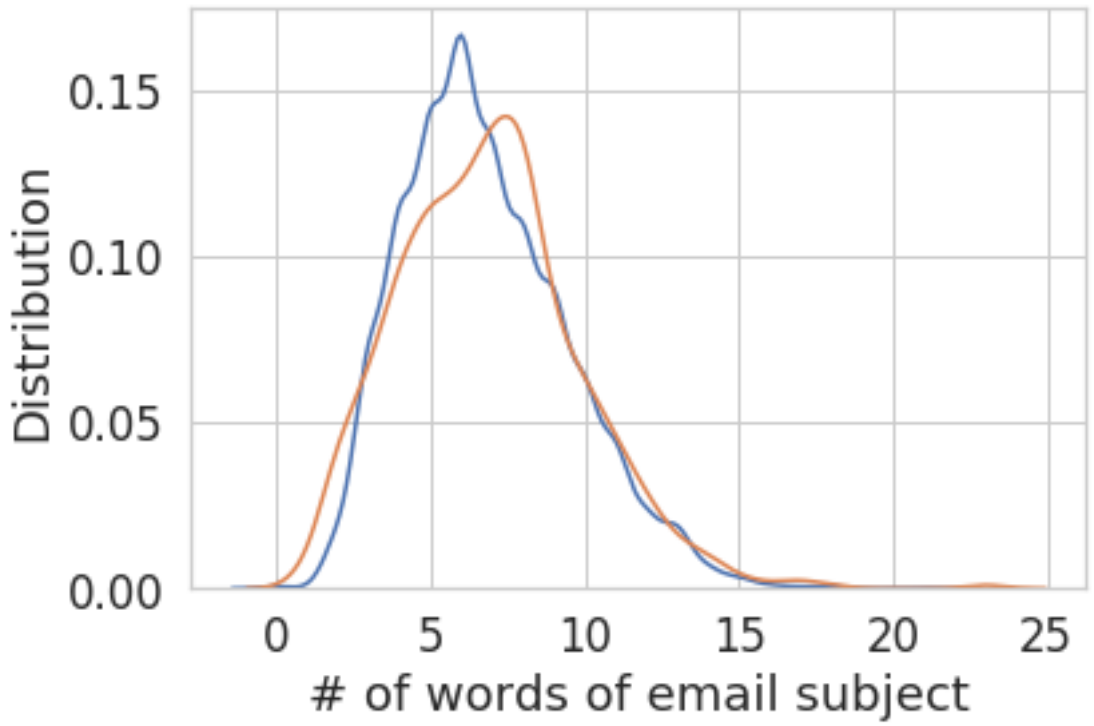
[29]:
```
#Testing length of subject
length = train['subject'].apply(len)
tempDataFrame = pd.DataFrame({
    'spam': train['spam'],
    'length': length
})
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 0]['length'], hist = None,␣
↪label = 'Ham')
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 1]['length'], hist = None,␣
↪label = 'Spam')
plt.xlabel('Length of email subject')

plt.ylabel('Distribution')
plt.savefig('training_conditional_densities.png')
```



[30]:
```
# Testing exclamation mark counts
length = train['email'].apply(lambda x: x.count('!'))
tempDataFrame = pd.DataFrame({
    'spam': train['spam'],
    'length': length
})
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 0]['length'], hist = None,␣
 ↪label = 'Ham')
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 1]['length'], hist = None,␣
 ↪label = 'Spam')
plt.xlabel('# of ! of email body')
plt.xlim([0,50])

plt.ylabel('Distribution')
plt.savefig('training_conditional_densities.png')
```



[31]:
```
#Testing word length
length = train['subject'].apply(lambda x: x.split()).apply(len)
tempDataFrame = pd.DataFrame({
    'spam': train['spam'],
    'length': length
})
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 0]['length'], hist = None,␣
 ↪label = 'Ham')
sns.distplot(tempDataFrame[tempDataFrame['spam'] == 1]['length'], hist = None,␣
 ↪label = 'Spam')
plt.xlabel('# of words of email subject')

plt.ylabel('Distribution')
plt.savefig('training_conditional_densities.png')
```



[32]:
```
emailWords = ['<html>', 'money','url', 'business', 'click', 'subscribe',␣
↪'daily', 'win', 'buy', 'million', 'shipping', '!', 'earn', 'opportunity',␣
↪'cash', 'credit']
emailMatrix = words_in_texts(emailWords, test['email'])
subjectWords = ['RE:']
subjectMatrix = words_in_texts(subjectWords, test['subject'].fillna(''))
htmlSyntaxMatrix = getPairedHTMLTagsMatrix(test['email'])
finalMatrix = []

for i in range(len(emailMatrix)):
    finalMatrix.append(emailMatrix[i] + subjectMatrix[i] + htmlSyntaxMatrix[i])

test_predictions = model.predict(finalMatrix)
```

[33]:
```
grader.check("q10")
```

[33]:
```
    All tests passed!
```

The following cell generates a CSV file with your predictions. **You must submit this CSV file to the "Project 2 Test Predictions" assignment on Gradescope to get credit for this question.**

[34]:
```
from datetime import datetime

# Assuming that your predictions on the test set are stored in a 1-dimensional␣
 ↪array called
# test_predictions. Feel free to modify this cell as long you create a CSV in␣
 ↪the right format.

# Construct and save the submission:
submission_df = pd.DataFrame({
    "Id": test['id'],
    "Class": test_predictions,
}, columns=['Id', 'Class'])
timestamp = datetime.isoformat(datetime.now()).split(".")[0]
submission_df.to_csv("submission_{}.csv".format(timestamp), index=False)

print('Created a CSV file: {}.'.format("submission_{}.csv".format(timestamp)))
print('You may now upload this CSV file to Gradescope for scoring.')
```

```
Created a CSV file: submission_2020-11-30T13:58:36.csv.
You may now upload this CSV file to Gradescope for scoring.
```

To double-check your work, the cell below will rerun all of the autograder tests.

[35]:
```
grader.check_all()
```

[35]:
```
q10:

    All tests passed!


q1a:

    All tests passed!


q1b:

    All tests passed!


q2:

    All tests passed!


q4:
```

```
    All tests passed!
```

```
q5:

    All tests passed!
```

```
q6a:

    All tests passed!
```

```
q6b:

    All tests passed!
```

```
q6d:

    All tests passed!
```

7.1 Submission

Make sure you have run all cells in your notebook in order before running the cell below, so that all images/graphs appear in the output. The cell below will generate a zip file for you to submit. **Please save before exporting!**

[36]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export("proj2.ipynb")
```

```
<IPython.core.display.HTML object>
```

[ ]:

[ ]:

[ ]:

[ ]: