Project 2: Spam/Ham Classification

Feature Engineering, Logistic Regression, Cross Validation

Due Date: Tuesday 8/6/19, 11:59PM

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*

This Assignment

In this project, you will use what you've learned in class to create a classifier that can distinguish spam (junk or commercial or bulk) emails from ham (non-spam) emails. In addition to providing some skeleton code to fill in, we will evaluate your work based on your model's accuracy and your written responses in this notebook.

After this project, you should feel comfortable with the following:

- Feature engineering with text data
- Using sklearn libraries to process data and fit models
- Validating the performance of your model and minimizing overfitting
- Generating and analyzing precision-recall curves

Warning

We've tried our best to filter the data for anything blatantly offensive as best as we can, but unfortunately there may still be some examples you may find in poor taste. If you encounter these examples and believe it is inappropriate for students, please let a TA know and we will try to remove it for future semesters. Thanks for your understanding!

Score Breakdown

Question	Points
1a	1
1b	1
1c	2
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

Part I - Initial Analysis

In [1]:

```
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)
```

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 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 Kaggle for evaluation.

In [2]:

```
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: Tue Jul 30 14:59:41 2019
MD5 hash of file: 0380c4cf72746622947b9ca5db9b8be8
Using version already downloaded: Tue Jul 30 14:59:43 2019
MD5 hash of file: a2e7abd8c7d9abf6e6fafc1d1f9ee6bf
```

Out[2]:

	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

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.

```
BEGIN QUESTION
name: q1a
points: 1
```

```
In [3]:
```

```
# BEGIN SOLUTION
print('Before imputation:')
print(original_training_data.isnull().sum())
original_training_data = original_training_data.fillna('')
print('------------')
print('After imputation:')
print(original_training_data.isnull().sum())
# END SOLUTION
```

```
Before imputation:
id          0
subject     6
email       0
spam        0
dtype: int64
------------
After imputation:
id          0
subject     0
email       0
spam        0
dtype: int64
```

```
In [4]:
```

```
# TEST
original_training_data.isnull().sum().sum()
```

```
Out[4]:

0
```

Question 1b

In the cell below, print the text of 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.

```
BEGIN QUESTION
name: q1b
points: 1
```

In [5]:

```
first_ham = original_training_data.loc[original_training_data['spam'] == 0, 'ema
il'].iloc[0] # SOLUTION
first_spam = original_training_data.loc[original_training_data['spam'] == 1, 'em
ail'].iloc[0] # SOLUTION
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 i
ts tent due
 to the bankruptcy of its parent company. a&l daily will be auctione
d 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 o
r fall =
 short. it's totally up<br>to you. our methods are guaranteed to inc
rease 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>
```

In [6]:

```
# TEST
len(first_ham) > 0 and first_ham[:0] == ''
```

Out[6]:

```
True
```

In [7]:

```
# TEST
len(first_spam) > 0 and first_spam[:0] == ''
```

Out[7]:

```
True
```

In [8]:

```
# HIDDEN TEST
original_training_data.loc[original_training_data['spam'] == 0, 'email'].iloc[0]
in first_ham
```

Out[8]:

```
True
```

In [9]:

```
# HIDDEN TEST
original_training_data.loc[original_training_data['spam'] == 1, 'email'].iloc[0]
in first_spam
```

Out[9]:

```
True
```

Question 1c

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

```
BEGIN QUESTION
name: q1c
manual: True
points: 2
```

SOLUTION:

It looks like the spam email has HTML tags. If many spam emails have HTML tags, we can use them to predict whether an email is spam or ham.

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.

In [10]:

```
from sklearn.model_selection import train_test_split

train, val = train_test_split(original_training_data, test_size=0.1, random_stat
e=42)
```

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 XX and a vector of corresponding binary labels yy. 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 XX is an email. Each column of XX contains one feature for all the emails. We'll guide you through creating a simple feature, and you'll create more interesting ones when you are trying to increase your accuracy.

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.

```
BEGIN QUESTION
name: q2
points: 3
```

In [11]:

```
def words_in_texts(words, texts):
    '''
    Args:
        words (list-like): 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.
    '''
    indicator_array = 1 * np.array([texts.str.contains(word) for word in words])
.T # SOLUTION
    return indicator_array
```

In [12]:

```
# TEST
np.allclose(words_in_texts(['hello', 'bye', 'world'],
                           pd.Series(['hello', 'hello worldhello'])),
            np.array([[1, 0, 0],
                      [1, 0, 1]]))
```

Out[12]:

```
True
```

In [13]:

```
# TEST
np.allclose(words_in_texts(['a', 'b', 'c', 'd', 'e', 'f', 'g'],
                           pd.Series(['a b c d ef g', 'a', 'b', 'c', 'd e f g',
'h', 'a h'])),
            np.array([[1,1,1,1,1,1,1],
                      [1,0,0,0,0,0,0],
                      [0,1,0,0,0,0,0],
                      [0,0,1,0,0,0,0],
                      [0,0,0,1,1,1,1],
                      [0,0,0,0,0,0,0],
                      [1,0,0,0,0,0,0]]))
```

Out[13]:

```
True
```

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.

training conditional proportions

Hint:

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

In [14]:

```
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 some words column and a type colu
mn. You can think of each row is a sentence, and the value of 1 or 0 indicates t
he number of occurances of the word in this sentence."))
display(df);
display(Markdown("> `melt` will turn columns into variale, notice how `word_1` a
nd `word_2` become `variable`, their values are stoed in the value column"))
display(df.melt("type"))
```

Our Original DataFrame has some words column and a type column. You can think of each row is a sentence, and the value of 1 or 0 indicates the number of occurances 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 variale, notice how `word_1` and `word_2` become `variable`, their values are stoed 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

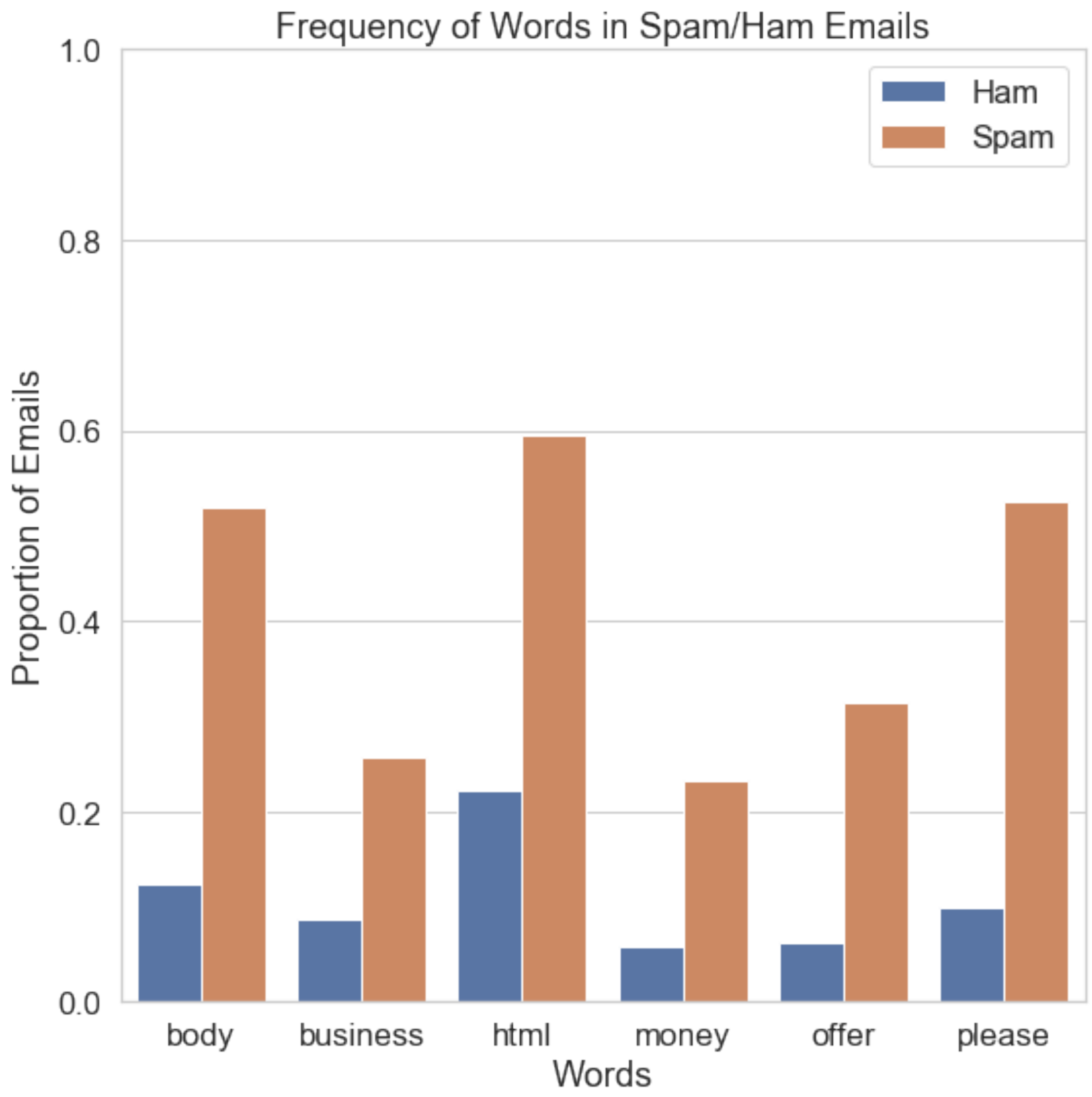
Question 3a

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`.

```
BEGIN QUESTION
name: q3a
manual: True
format: image
points: 2
```

In [15]:

```
train=train.reset_index(drop=True) # We must do this in order to preserve the or
dering of emails to labels for words_in_texts

# BEGIN SOLUTION
some_words = ['body', 'html', 'please', 'money', 'business', 'offer']
Phi_train = words_in_texts(some_words, train['email'])

df = pd.DataFrame(data = Phi_train, columns = some_words)
df['label'] = train['spam']

plt.figure(figsize=(8,8))
sns.barplot(x = "variable",
            y = "value",
            hue = "label",
            data = (df
                    .replace({'label':
                                {0 : 'Ham',
                                 1 : 'Spam'}})
                    .melt('label')
                    .groupby(['label', 'variable'])
                    .mean()
                    .reset_index()))

plt.ylim([0, 1])
plt.xlabel('Words')
plt.ylabel('Proportion of Emails')
plt.legend(title = "")
plt.title("Frequency of Words in Spam/Ham Emails")
plt.tight_layout()
# plt.savefig("images/training_conditional_proportions.png")
plt.show()
# END SOLUTION
```



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.

training conditional densities

Question 3b

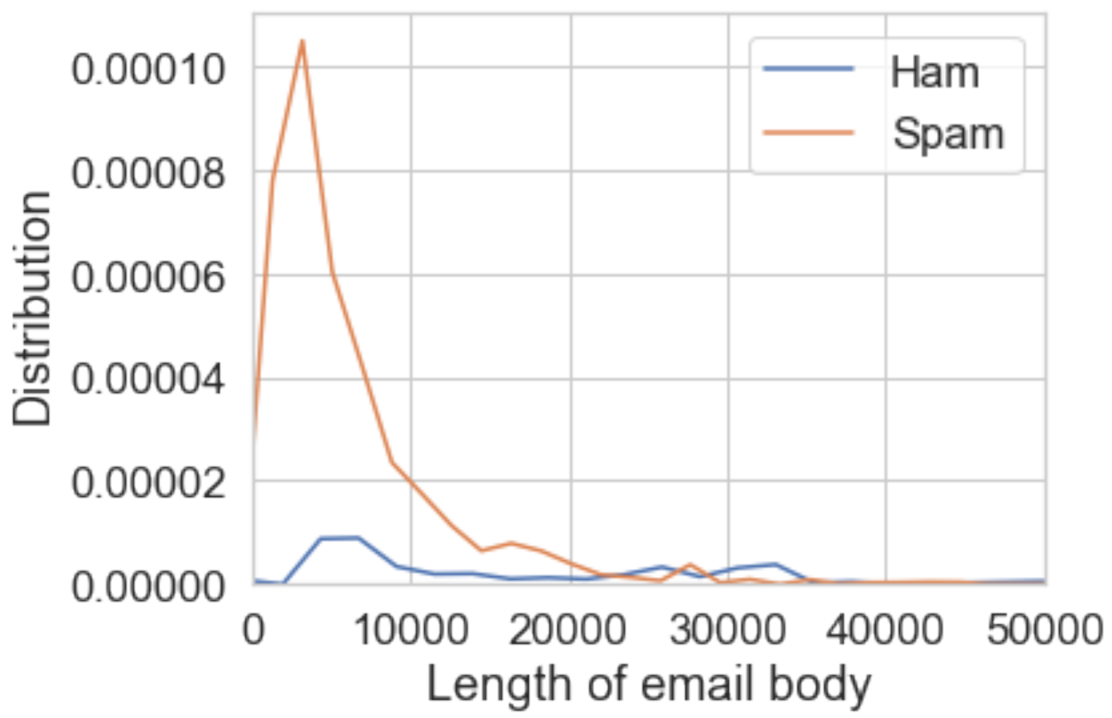
Create a *class conditional density plot* like the one above (using `sns.distplot`), comparing the distribution of the length of spam emails to the distribution of the length of ham emails in the training set. Set the x-axis limit from 0 to 50000.

```
BEGIN QUESTION
name: q3b
manual: True
format: image
points: 2
```

In [16]:

```
# BEGIN SOLUTION
tmp = train.copy()
tmp['length'] = tmp['email'].str.len()
sns.distplot(tmp.loc[tmp['spam'] == 0, 'length'],hist=False, label='Ham')
sns.distplot(tmp.loc[tmp['spam'] == 1, 'length'],hist=False, label='Spam')
plt.xlabel('Length of email body')
plt.ylabel('Distribution')
plt.xlim((0,50000))
plt.tight_layout()
# plt.savefig("images/training_conditional_densities2.png")
# END SOLUTION
```

```
/Users/raguvirkunani/anaconda3/lib/python3.7/site-packages/scipy/sta
ts/stats.py:1713: FutureWarning: Using a non-tuple sequence for mult
idimensional indexing is deprecated; use `arr[tuple(seq)]` instead o
f `arr[seq]`. In the future this will be interpreted as an array ind
ex, `arr[np.array(seq)]`, which will result either in an error or a
different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumva
l
```



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!

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 0 and 1, no other values). It does not check that your function is correct; that was verified in a previous question.

```
BEGIN QUESTION
name: q4
points: 2
```

In [17]:

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

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

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

Out[17]:

```
(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]))
```

In [18]:

```
# TEST
X_train.shape
```

Out[18]:

```
(7513, 5)
```

In [19]:

```
# TEST
np.unique(X_train) # X matrix should consist of only 0 or 1
```

Out[19]:

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

In [20]:

```
# TEST
np.unique(Y_train) # y vector should consist of only 0 or 1
```

Out[20]:

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

Question 5

Now we have matrices we can give to scikit-learn! Using the `LogisticRegression` (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) classifier, train a logistic regression model using `X_train` and `Y_train`. Then, output the accuracy of the model (on the training data) in the cell below. You should get an accuracy around 0.75.

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

```
BEGIN QUESTION
name: q5
points: 2
```

In [21]:

```
from sklearn.linear_model import LogisticRegression

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

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

```
Training Accuracy:  0.7576201251164648
```

In [22]:

```
# TEST
training_accuracy > 0.72
```

Out[22]:

```
True
```

Evaluating Classifiers

That doesn't seem too shabby! But the classifier you made above isn't as good as this might lead us to believe. First, we are evaluating accuracy on the training set, which may lead to a misleading accuracy measure, especially if we used the training set to identify discriminative features. In future parts of this analysis, it will be safer to 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.

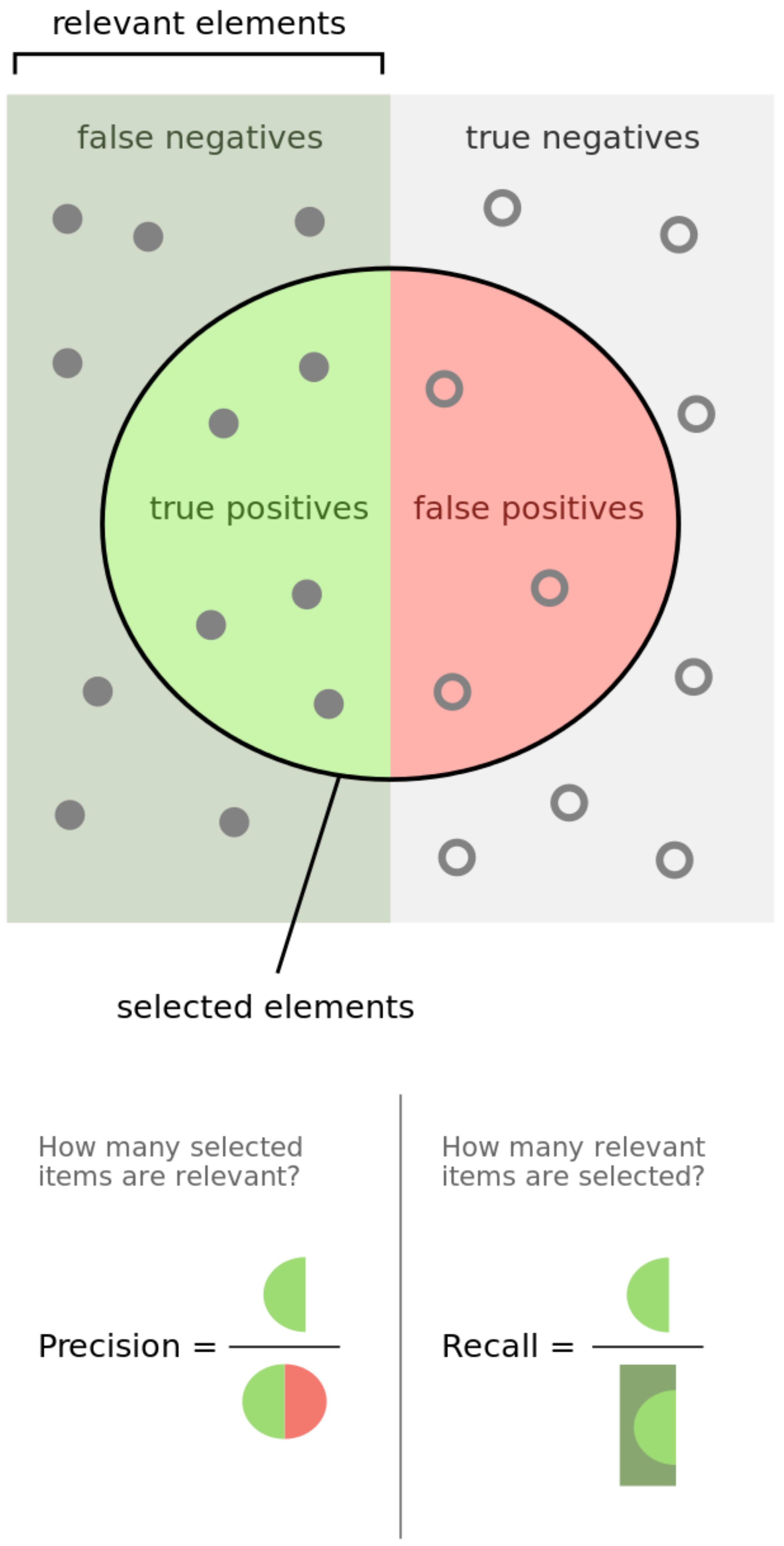
These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

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

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

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

The following image might help:


relevant elements
false negatives
true negatives
true positives
false positives
selected elements
How many selected items are relevant?
How many relevant items are selected?
Precision =
Recall =

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.

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 (answers can be hard-coded):

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.

```
BEGIN QUESTION
name: q6a
points: 1
```

In [23]:

```
zero_predictor_fp = 0 # SOLUTION
zero_predictor_fn = sum(Y_train == 1) # SOLUTION
```

In [24]:

```
# TEST
zero_predictor_fp >= 0
```

Out[24]:

```
True
```

In [25]:

```
# TEST
zero_predictor_fn >= 0
```

Out[25]:

```
True
```

In [26]:

```
# HIDDEN TEST
zero_predictor_fp
```

Out[26]:

```
0
```

```
In [27]:
```

```
# HIDDEN TEST
zero_predictor_fn
```

```
Out[27]:

1918
```

Question 6b

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

```
BEGIN QUESTION
name: q6b
points: 1
```

```
In [28]:
```

```
zero_predictor_acc = np.mean(Y_train == 0) # SOLUTION
zero_predictor_recall = 0 # SOLUTION
```

```
In [29]:
```

```
# TEST
zero_predictor_acc >= 0
```

```
Out[29]:

True
```

```
In [30]:
```

```
# TEST
zero_predictor_recall >= 0
```

```
Out[30]:

True
```

```
In [31]:
```

```
# HIDDEN TEST
np.isclose(zero_predictor_acc, 0.7447091707706642)
```

```
Out[31]:

True
```

In [32]:

```
# HIDDEN TEST
zero_predictor_recall
```

Out[32]:

```
0
```

Question 6c

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

```
BEGIN QUESTION
name: q6c
manual: True
points: 2
```

SOLUTION:

- There are no false positives ($\text{FP} = 0$) because nothing is labeled spam.
- Every spam email is mislabeled as ham, so the number of false negatives is equal to the number of spam emails in the training data ($\text{FN} = 1918$).
- The classifier correctly labels 74.47% of observations in the training data.
- The classifier recalls none (0%) of the spam observations.

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.

```
BEGIN QUESTION
name: q6d
points: 2
```

In [33]:

```
# BEGIN SOLUTION NO PROMPT
Y_train_hat = model.predict(X_train)

TP = sum((Y_train_hat == Y_train) & (Y_train_hat == 1))
TN = sum((Y_train_hat == Y_train) & (Y_train_hat == 0))
FP = sum((Y_train_hat != Y_train) & (Y_train_hat == 1))
FN = sum((Y_train_hat != Y_train) & (Y_train_hat == 0))
# END SOLUTION
logistic_predictor_precision = TP / (TP + FP) # SOLUTION
logistic_predictor_recall = TP / (TP + FN) # SOLUTION
logistic_predictor_far = FP / (FP + TN) # SOLUTION
```

In [34]:

```
# TEST
logistic_predictor_precision >= 0
```

Out[34]:

```
True
```

In [35]:

```
# TEST
logistic_predictor_recall >= 0
```

Out[35]:

```
True
```

In [36]:

```
# TEST
logistic_predictor_far >= 0
```

Out[36]:

```
True
```

In [37]:

```
# HIDDEN TEST
np.isclose(logistic_predictor_precision, 0.6422287390029325)
```

Out[37]:

```
True
```

In [38]:

```
# HIDDEN TEST
np.isclose(logistic_predictor_recall, 0.11418143899895725)
```

Out[38]:

```
True
```

In [39]:

```
# HIDDEN TEST
np.isclose(logistic_predictor_far, 0.021805183199285077)
```

Out[39]:

```
True
```

Question 6e

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

```
BEGIN QUESTION
name: q6e
manual: True
points: 1
```

SOLUTION:

There are more false negatives ($\mathrm{FN} = 1699$) than false positives ($\mathrm{FP} = 122$).

Question 6f

1. Our logistic regression classifier got 75.6% 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.

```
BEGIN QUESTION
name: q6f
manual: True
points: 3
```

SOLUTION:

1. An accuracy of 75% means that we're only doing slightly better than guessing ham for every email.
2. One reason why our classifier isn't great is that the matrix `X_train` has many rows with all 0. That is, the words we've chosen as our features aren't actually present in many of the emails so the classifier can't use them to distinguish between ham/spam emails.
3. The false-alarm rate for logistic regression is too high, even at 2%: ideally false-alarms would almost never happen. I might rather wade through thousands of spam emails than get 2% of legitimate emails filtered out of my inbox.

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 Kaggle.

Kaggle 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 the entire set when submitting to Kaggle (the accuracy that will determine your score for question 10).

Here are some ideas for improving your model:

1. Finding better features based on the email text. Some example features are:
   A. Number of characters in the subject / body
   B. Number of words in the subject / body
   C. Use of punctuation (e.g., how many '!' were there?)
   D. Number / percentage of capital letters
   E. Whether the email is a reply to an earlier email or a forwarded email
2. Finding better 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.

ou 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 should use the **validation data** to evaluate your model and get a better sense of how it will perform on the Kaggle evaluation.*

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 / didn't work?
3. What was surprising in your search for good features?

```
BEGIN QUESTION
name: q7
manual: True
points: 6
```

SOLUTION TODO

Question 8: EDA

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

1. A plot showing something meaningful about the data that helped you during feature / model selection.
2. 2-3 sentences describing what you plotted and what its implications are for your features.

Feel to create as many plots as you want in your process of feature selection, but select 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.

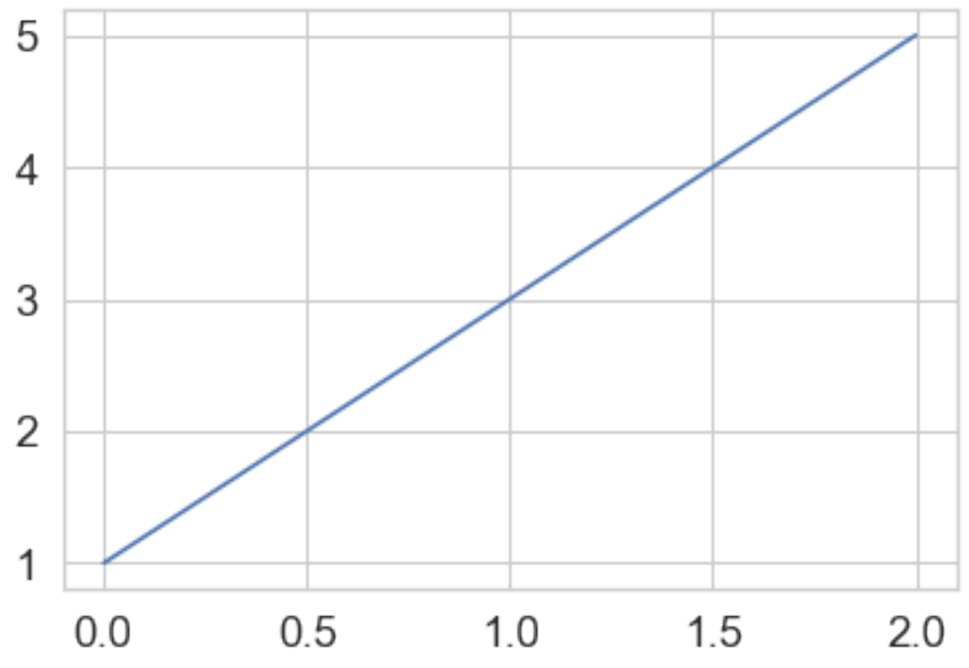
```
BEGIN QUESTION
name: q8
manual: True
format: image
points: 6
```

In [40]:

```
# Write your description (2-3 sentences) as a comment here:
#
#
#

# Write the code to generate your visualization here:
# BEGIN SOLUTION
plt.plot([1, 3, 5]) # This is a dummy plot, not a real example of a solution
# END SOLUTION
```

Out[40]:

```
[<matplotlib.lines.Line2D at 0x1a2306b2b0>]
```



Question 9: ROC Curve

In most cases we won't be able to get no false positives and no 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 a disease until it's too late to treat, while a false positive means that a patient will probably have to take another screening.

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 ≥ 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 ≥ 0.7 probability of being spam, for example. This is how we can trade off false positives and false negatives.

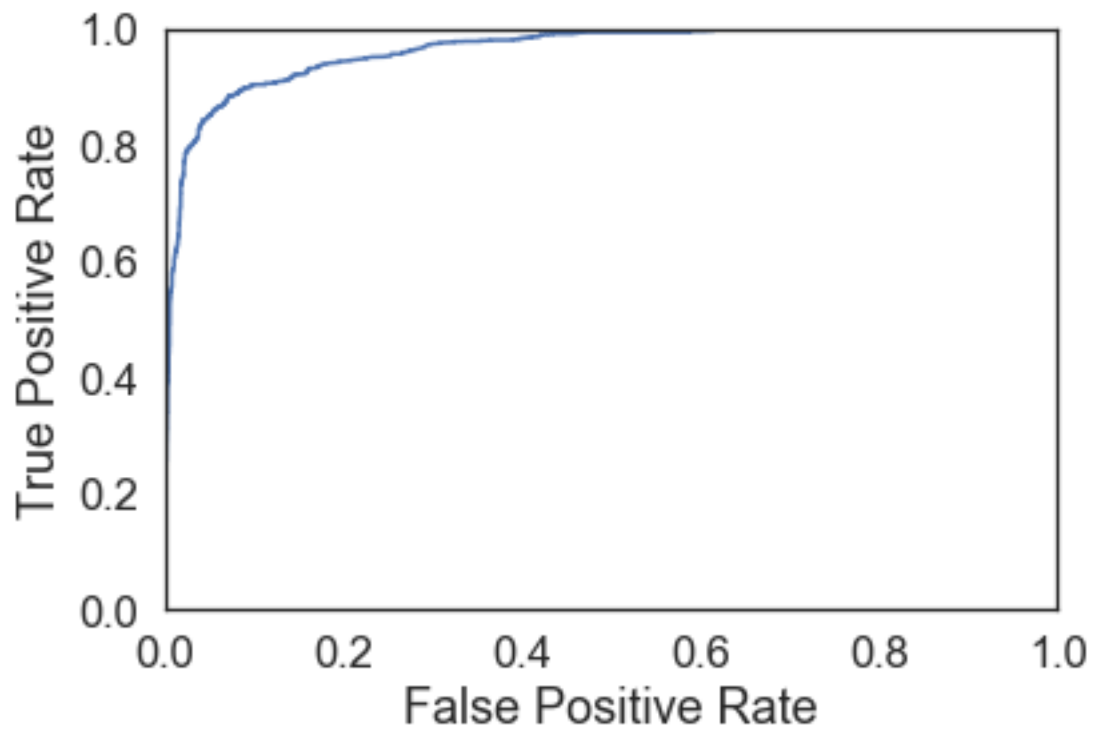
The ROC curve shows this trade off for each possible cutoff probability. In the cell below, plot an ROC curve for your final classifier (the one you use to make predictions for Kaggle). Refer to the Lecture 20 notebook to see how to plot an ROC curve.

```
BEGIN QUESTION
name: q9
manual: True
points: 3
```

In [41]:

```
from sklearn.metrics import roc_curve

# Note that you'll want to use the .predict_proba(...) method for your classifie
r
# instead of .predict(...) so you get probabilities, not classes

# BEGIN SOLUTION
staff_words = ['body', 'click', 'please', 'base64', '2002', 'html', 'subscribed'
,
               'wrote', 'mortgage', 'align3dcenterfont', 'dear', 'br', 'width10i
mg',
               'divfont', 'im', 'receive', 'list', 'tags', 'web', 'base64', 'cli
ck',
               'body', 'please', 'money', 'offer', 'receive', 'contact', 'free',
               'tr', 'removed', 'remove', 'html', 'font', 'form',
               'credit', 'business', 'div']

X_train = words_in_texts(staff_words, train['email'])

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

print('accuracy: ', staff_model.score(X_train, Y_train))

Y_predict = staff_model.predict_proba(X_train)[:, 1]
fpr, tpr, thresholds = roc_curve(Y_train, Y_predict)
with sns.axes_style("white"):
    plt.plot(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.show()
# END SOLUTION
```

```
accuracy:  0.9245308132570211
```



Question 10: Submitting to Kaggle

The following code will write your predictions on the test dataset to a CSV, which you can submit to Kaggle. You may need to modify it to suit your needs.

Save your predictions in a 1-dimensional array called `test_predictions`. *Even if you are not submitting to Kaggle, please make sure you've saved your predictions to `test_predictions` as this is how 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.

You should submit your CSV files to https://www.kaggle.com/c/ds100su19 (https://www.kaggle.com/c/ds100su19)

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

```
BEGIN QUESTION
name: q10
points: 15
```

In [42]:

```
test_predictions = staff_model.predict(words_in_texts(staff_words, test['email']
)) # SOLUTION
```

In [43]:

```
# TEST
isinstance(test_predictions, np.ndarray) # must be ndarray of predictions
```

Out[43]:

```
True
```

In [44]:

```
# TEST
np.unique(test_predictions) # must be binary labels (0 or 1) and not probabiliti
es
```

Out[44]:

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

In [45]:

```
# TEST
test_predictions.shape # must be the right number of predictions
```

Out[45]:

```
(1000,)
```

In [46]:

```
# HIDDEN TEST
test_sol = pd.read_csv('kaggle_solutions.csv')
test_labels = test_sol['Class'].values
score = np.mean(test_labels == test_predictions)
print("Staff Model Score:", score)

score > 0
```

```
Staff Model Score: 0.902
```

Out[46]:

```
True
```

In [47]:

```
# HIDDEN TEST
score > 0.82
```

Out[47]:

```
True
```

In [48]:

```
# HIDDEN TEST
score > 0.85
```

Out[48]:

```
True
```

In [49]:

```
# HIDDEN TEST
score > 0.88
```

Out[49]:

```
True
```

The following saves a file to submit to Kaggle.

In [50]:

```
from datetime import datetime

# Assuming that your predictions on the test set are stored in a 1-dimensional a
rray called
# test_predictions. Feel free to modify this cell as long you create a CSV in th
e 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 Kaggle for scoring.')
```

```
Created a CSV file: submission_2019-07-31T15:05:44.csv.
You may now upload this CSV file to Kaggle for scoring.
```