proj2a

May 25, 2022

[1]:
```
# Initialize Otter
import otter
grader = otter.Notebook("proj2a.ipynb")
```

1 Project 2A: Spam/Ham Classification

1.1 Feature Engineering, Logistic Regression

1.2 Due Date: Thursday April 21, 11:59PM PDT

Collaboration Policy

Data science is a collaborative activity. While you may talk with others about the homework, 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 This Assignment

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 homework, 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

This first part of the project focuses on initial analysis. In the second part of this project (to be released next week), you will build your own spam/ham classifier.

1.4 Warning

This is a **real world** dataset – the emails you are trying to classify are actual spam and legitimate emails. As a result, some of the spam emails may be in poor taste or be considered inappropriate. We think the benefit of working with realistic data outweighs these innapropriate emails, and wanted to give a warning at the beginning of the homework so that you are made aware.

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

# more readable exceptions
%pip install --quiet iwut
%load_ext iwut
%wut on
```

```
Note: you may need to restart the kernel to use updated packages.
```

1.5 Score Breakdown

Question	Points
1	2
2	3
3	3
4	2
5	2
6a	1
6b	1
6c	2
6d	2
6e	1
6f	3
Total	22

2 Part 1: 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 is from SpamAssassin. It 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.

Note: The dataset is from 2004, so the contents of emails might be very different from those in 2022.

Run the following cells to load 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]:
```
import zipfile
with zipfile.ZipFile('spam_ham_data.zip') as item:
    item.extractall()
```

[5]:
```
original_training_data = pd.read_csv('train.csv')
test = pd.read_csv('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()
```

[5]:

	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

First, let's check if our data contains any missing values. We have filled in the cell below to print the number of NaN values in each column. If there are NaN values, we replace them with appropriate filler values (i.e., NaN values in the `subject` or `email` columns will be replaced with empty strings). Finally, we 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.

[6]:
```
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())
```

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

2.0.2 Question 1

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

[7]:
```
first_ham = original_training_data.loc[original_training_data['spam'] == 0,␣
↪'email'].iloc[0]
first_spam = original_training_data.loc[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>
```

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

As depicted in the output, the spam email utilizes HTML while the first email utilizes plain text.

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 random seed in the following questions, as our tests depend on it.**

[8]:
```
# 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 Part 2: 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 a 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.

[9]:
```
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.
    '''
    indicator_array = [texts.str.contains(x) for x in words]
    indicator_array = np.array(indicator_array)
    return indicator_array.T
```

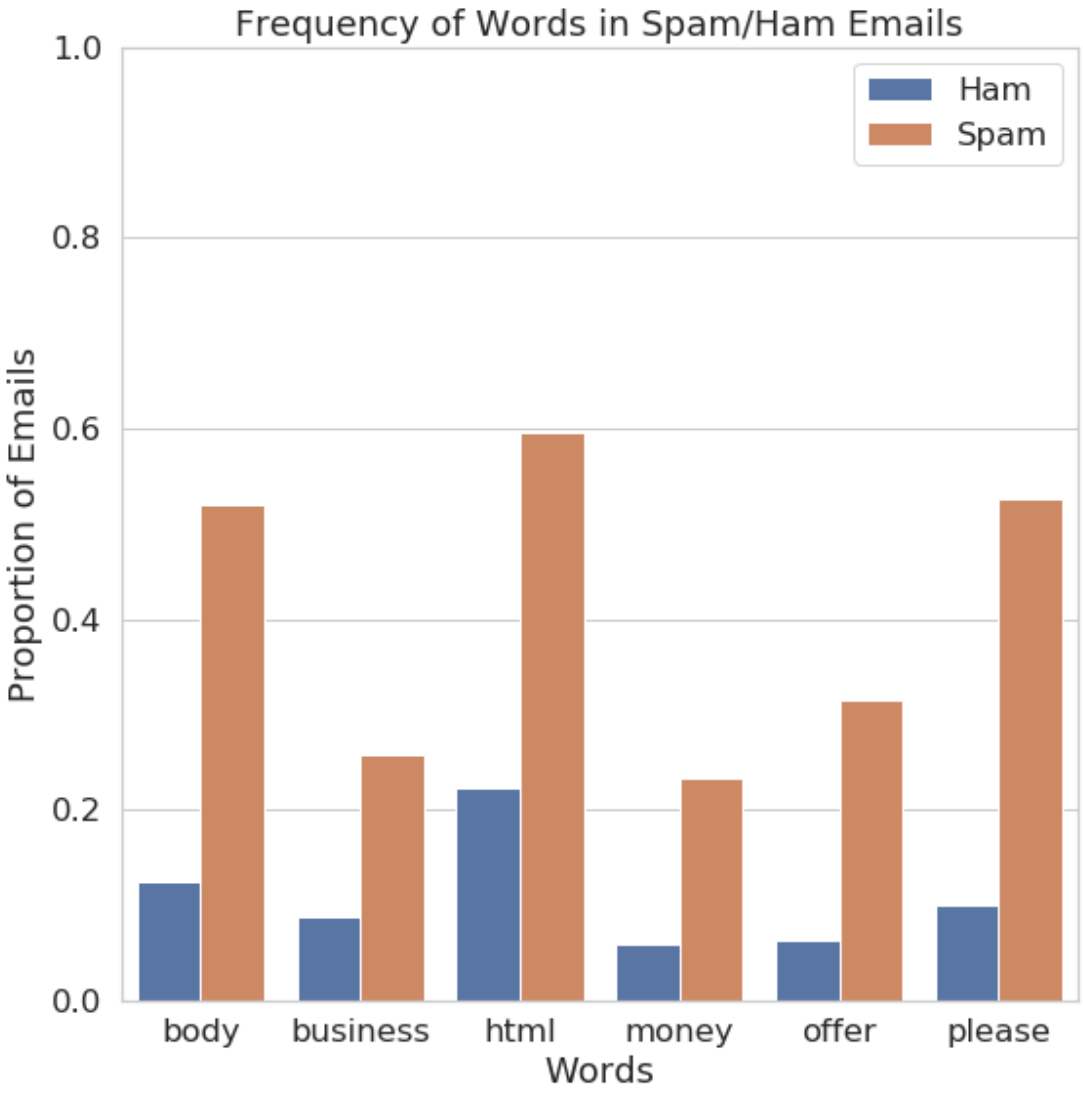
[10]:
```
grader.check("q2")
```

[10]:
```
q2 results: All test cases 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.

[11]:
```
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

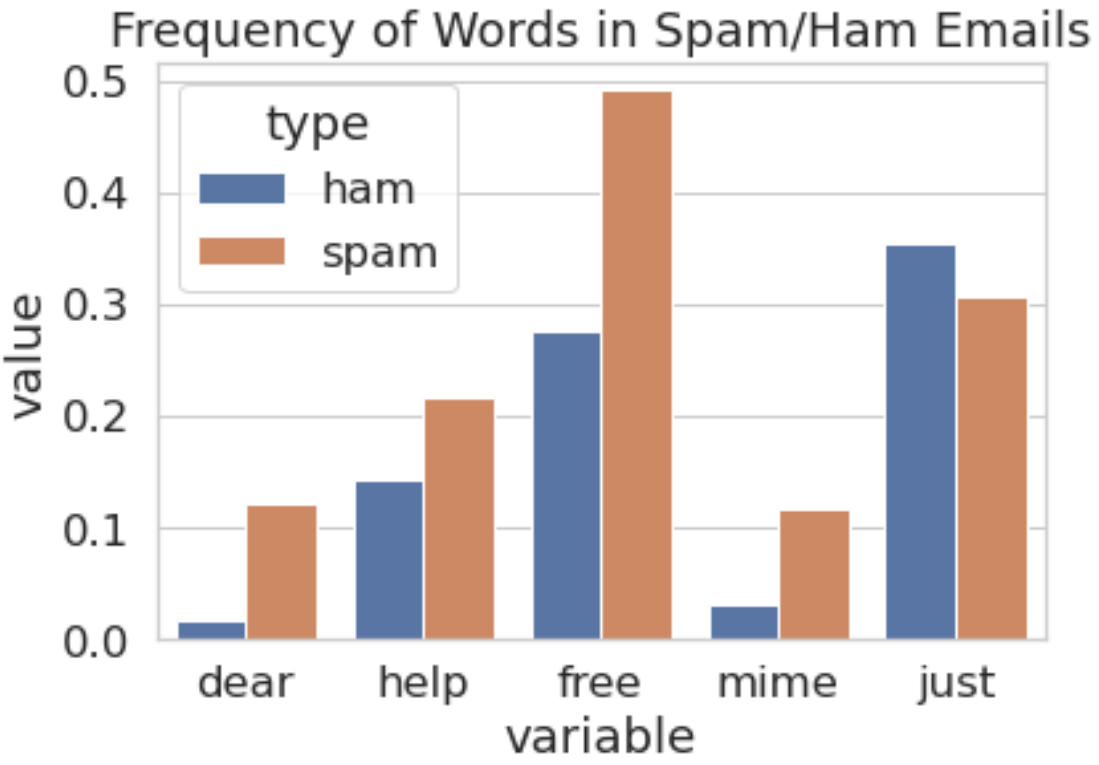
4.0.1 Question 3

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`.

[18]:
```
train = train.reset_index(drop=True) # We must do this in order to preserve the ordering of emails to labels for words_in_texts

words_int = ["dear", "help", "free", "mime", "just"]
selected_words = words_in_texts(words_int, train['email'])
matrix = np.matrix(selected_words)
tbl = pd.DataFrame(matrix).rename(columns = {0: "dear", 1: "help", 2: "free", 3: "mime", 4: "just"})
tbl["type"] = train["spam"]
tbl = tbl.melt("type")
tbl["type"] = tbl["type"].map({0: "ham", 1: "spam"})
sns.barplot(x = "variable", y = "value", hue = "type", data = tbl, ci = None)
plt.title("Frequency of Words in Spam/Ham Emails")
```

[18]:
```
Text(0.5, 1.0, 'Frequency of Words in Spam/Ham Emails')
```



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.

5 Part 3: 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.

[19]:
```
some_words = ['drug', 'bank', 'prescription', 'memo', 'private']

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

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

[19]:
```
(array([[False, False, False, False, False],
        [False, False, False, False, False],
        [False, False, False, False, False],
        [False, False, False, False, False],
        [False, False, False,  True, False]]),
 array([0, 0, 0, 0, 0]))
```

[20]:
```
grader.check("q4")
```

[20]:
```
q4 results: All test cases passed!
```

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.

[21]:
```
from sklearn.linear_model import LogisticRegression

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

training_accuracy = sum(predicted_y == Y_train) / len(Y_train)
print("Training Accuracy: ", training_accuracy)
```

```
Training Accuracy:  0.7576201251164648
```

[22]:
```
grader.check("q5")
```

[22]:
```
q5 results: All test cases passed!
```

6 Part 4: 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.

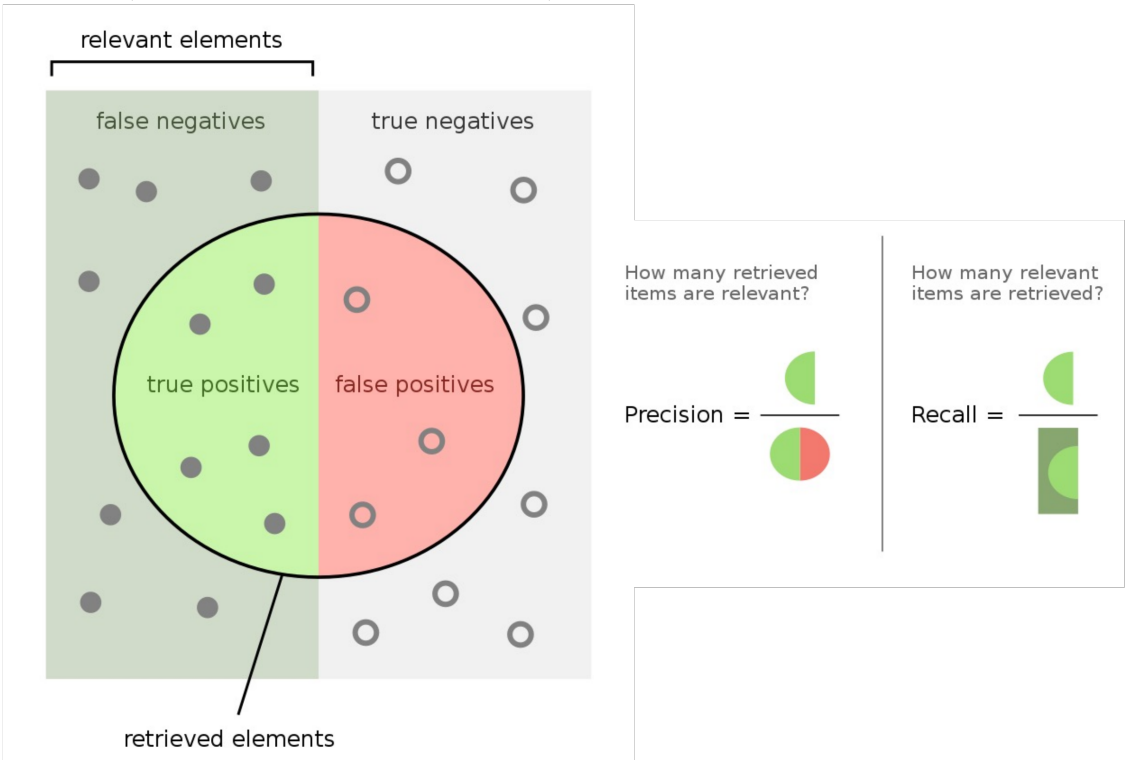
To be clear, we label spam emails as 1 and ham emails as 0. These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

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 below graphic (modified slightly from Wikipedia) 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.

6.0.1 Question 6

6.0.2 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.

[23]:
```
zero_predictor_fp = 0
zero_predictor_fn = Y_train.sum()
zero_predictor_fp, zero_predictor_fn
```

[23]:
```
(0, 1918)
```

[24]:
```
grader.check("q6a")
```

[24]:
```
q6a results: All test cases passed!
```

6.0.3 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.

[25]:
```
zero_predictor_acc = (len(Y_train) - Y_train.sum()) / len(Y_train)
zero_predictor_recall = 0
zero_predictor_acc, zero_predictor_recall
```

[25]:
```
(0.7447091707706642, 0)
```

[26]:
```
grader.check("q6b")
```

[26]:
```
q6b results: All test cases passed!
```

6.0.4 Question 6c

Comment on the results from 6a and 6b. For **each** of FP, FN, accuracy, and recall, briefly explain why we see the result that we do.

Since the zero predictor will always predict 0, we know the false positive will also be 0. For the FN, the false negative is 1918 which means there are 1918 spam emails which are wrongly classified as ham emails. For accuracy, it is estimated to be around 0.75 which demonstrates the proportion of how many ham emails are accurately predicted as ham emails. And for recall, we know the zero predictor will predict 0, so the true positive is 0, meaning the recall is also 0.

6.0.5 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, with the exception of the `.predict` method of your model object.

[27]:
```
logistic_predictor_precision = sum((Y_train == 1) & (predicted_y == 1)) /␣
↪sum((predicted_y == 1))
logistic_predictor_recall = sum((Y_train == 1) & (predicted_y == 1)) /␣
↪sum((Y_train == 1))
logistic_predictor_far = sum((Y_train == 0) & (predicted_y == 1)) /␣
↪sum((Y_train == 0))
```

[28]:
```
grader.check("q6d")
```

[28]:
```
q6d results: All test cases passed!
```

6.0.6 Question 6e

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

There are less false positives and more false negatives in the question above, in comparison to question 5.

6.0.7 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.

1. The prediction accuracy is on the lower end because it's accuracy is similar to that of the zero_predictor which was around 74%.
2. Some of the words given in the array aren't stated in the emails as often as others, or in other words, some words skew the results of the classifier as they don't demons
3. Of the two classifiers, I would prefer the logistic regression classifier for a spam filter because it's training accuracy and recall was higher. Essentially, this means that more spam emails were accurately labeled as spam emails in this model.

6.1 Congratulations! You have finished Project 2A!

In Project 2B, you will focus on building a spam/ham email classifier with logistic regression. You will be well-prepared to build such a model: you have considered what is in this data set, what it can be used for, and engineered some features that should be useful for prediction.

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

[29]:
```
grader.check_all()
```

[29]:
```
q2 results: All test cases passed!

q4 results: All test cases passed!

q5 results: All test cases passed!

q6a results: All test cases passed!

q6b results: All test cases passed!

q6d results: All test cases passed!
```

6.2 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!**

[30]:
```
# Save your notebook first, then run this cell to export your submission.
grader.export()
```

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