projB1

January 10, 2025

1 Project B1: Spam/Ham Classification

1.1 Introduction

You will use what you've learned in class to create a binary classifier that can distinguish spam (junk, commercial, or bulk) emails from ham (regular 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 the `sklearn` library to process data and fit models.
- Validating the performance of your model and minimizing overfitting.

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

1.2 Content 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 inappropriate emails but wanted to provide a warning at the beginning of the project so that you are 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.
```

[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 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 8,348 labeled examples, and the unlabeled test set contains 1,000 unlabeled examples.

Note: The dataset is from 2004, so the contents of emails might be very different from those in 2024.

Run the following cells to load the data into a `DataFrame`.

The `train` `DataFrame` contains labeled data you will use to train your model. It has 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 1,000 unlabeled emails. In Project B2, you will predict labels for these emails and submit your predictions to the autograder for evaluation.

[4]:
```
import zipfile

# Loading training and test datasets
with zipfile.ZipFile('spam_ham_data.zip') as item:
    with item.open("train.csv") as f:
        original_training_data = pd.read_csv(f)
    with item.open("test.csv") as f:
        test = pd.read_csv(f)
```

[5]:
```
# Convert the emails to lowercase as the first step of text processing.
original_training_data['email'] = original_training_data['email'].str.lower()
test['email'] = test['email'].str.lower()

original_training_data.head()
```

[5]:
```
   id                                            subject  \
0   0  Subject: A&L Daily to be auctioned in bankrupt…
1   1  Subject: Wired: "Stronger ties between ISPs an…
2   2  Subject: It's just too small                   …
```

```
3   3                      Subject: liberal defnitions\n
4   4  Subject: RE: [ILUG] Newbie seeks advice - Suse…

                                               email  spam
0  url: http://boingboing.net/#85534171\n date: n…     0
1  url: http://scriptingnews.userland.com/backiss…     0
2  <html>\n <head>\n </head>\n <body>\n <font siz…     1
3  depends on how much over spending vs. how much…     0
4  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: 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.

[6]:
```python
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
```

3 Part 1: Initial Analysis

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]:

```python
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("Ham Email:")
print(first_ham)
print("-------------------------------------------------")
print("Spam Email:")
print(first_spam)
```

```
Ham Email:
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



-------------------------------------------------
Spam Email:
<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>
```

3.1 Question 1

Discuss one attribute or characteristic you notice that is different between the two emails that may allow you to uniquely identify a spam email.

The spam email uses HTML tags and formatting such as 'html', 'body', 'head', 'font size' and so on, which are likely to be automated messages. In contrast, the ham email has a more natural and straightforward text.

3.2 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 datasets. 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)
```

4 Part 2: Feature Engineering

We want to take the text of an email and predict whether the email is ham or spam. This is a **binary 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 $\mathbb{X}$ 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 $\mathbb{X}$ is an email. Each column of $\mathbb{X}$ 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.

4.1 Question 2

Create a function `words_in_texts` that takes in a list of interesting words (`words`) and a `Series` of emails (`texts`). Our goal is to check if each word in `words` is contained in the emails in `texts`.

The `words_in_texts` function should output a **2-dimensional `NumPy` array** that contains one row for each email in `texts` and one column for each word in `words`. If the j-th word in `words` is present at least once in the i-th email in `texts`, the output array should have a value of 1 at the position (i, j). Otherwise, if the j-th word is not present in the i-th email, the value at (i, j) should be 0.

In Project B2, we will be applying `words_in_texts` to some large datasets, so implementing some form of vectorization (for example, using `NumPy` arrays, `Series.str` functions, etc.) is highly recommended. **You are allowed to use only *one* list comprehension or for loop**, and you should look into how you could combine that with the vectorized functions discussed above. **Do not use a double for loop, or you will run into issues later on in Project B2.**

For example:

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

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

Importantly, we **do not** calculate the *number of occurrences* of each word; only if the word is present at least *once*. Take a moment to work through the example on your own if need be — understanding what the function does is a critical first step in implementing it.

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:
        A 2D NumPy array of 0s and 1s with shape (n, d) where
        n is the number of texts, and d is the number of words.
    """
    indicator_array = np.array([texts.str.contains(word, regex=False).
  ↪astype(int) for word in words]).T
    return indicator_array
```

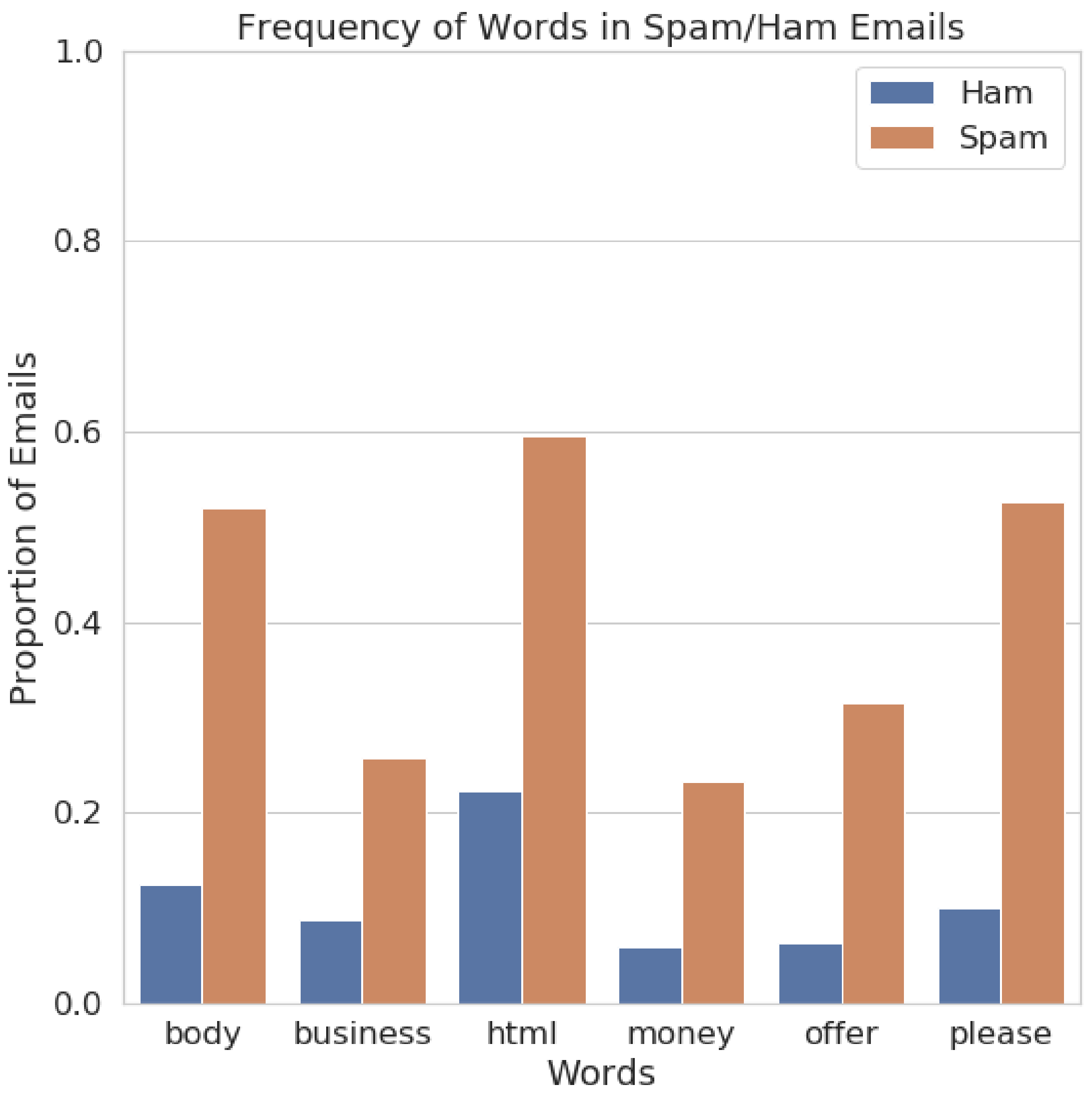
[10]:
```
# Run this cell to see what your function outputs. Compare the results to the␣
  ↪example provided above.
words_in_texts(['hello', 'bye', 'world'], pd.Series(['hello', 'hello␣
  ↪worldhello']))
```

[10]:
```
array([[1, 0, 0],
       [1, 0, 1]])
```

5 Part 3: 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. Suppose the feature is a binary indicator, such as whether a particular word occurs in the text. In that case, this compares the proportion of spam emails with the word to the proportion of ham emails with the word.

The following plot (created using `sns.barplot`) compares the proportion of emails in each class containing a particular set of words. The bars colored by email class were generated by setting the `hue` parameter of `sns.barplot` to a column containing the class (spam or ham) of each data point. An example of how this class column was created is shown below:



You can use `DataFrame`'s `.melt` (documentation) 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 occurrences 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 occurrences 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
```

5.1 Question 3

Create the bar chart above by comparing the proportion of spam and ham emails containing specific words. **Choose a set of 6 words other than those shown in the example.** These words should have different proportions for the two classes (i.e., noticeably different bar heights across spam and ham). Make sure only to consider emails from `train`. Your `words_in_texts` function from the previous part will be useful here.

Hint: This is a pretty challenging question. The suggested approach is to first look at the example bar plot and make sure you can interpret what is being plotted - what does a bar represent? What does the height mean?

Next, see how to make this plot with `sns.barplot`. Take a look at the documentation and determine what the inputs should be. A possible data input is given below:

```
  <th>type</th>      <th>variable</th>      <th>value</th>    </tr>  </thead>  <tbody>
```

0

```
  <td>Ham</td>      <td>word_1</td>      <td>0.021269</td>    </tr>
```

1

```
  <td>Ham</td>      <td>word_2</td>      <td>0.101519</td>    </tr>
```

2

```
  <td>Spam</td>      <td>word_3</td>      <td>0.059160</td>    </tr>
```

3

```
  <td>Spam</td>      <td>word_2</td>      <td>0.017694</td>    </tr>
```

4

```
  <td>Ham</td>      <td>word_4</td>      <td>0.013226</td>    </tr>
```

…

```
  <td>...</td>      <td>...</td>      <td>...</td>    </tr>
```

Finally, you will need to chain some `pandas` functions together. Try to add one function at a time and see how that affects the `DataFrame`. It may help to use a new cell or print out the `DataFrame` for debugging purposes as you work towards achieving the desired format above.

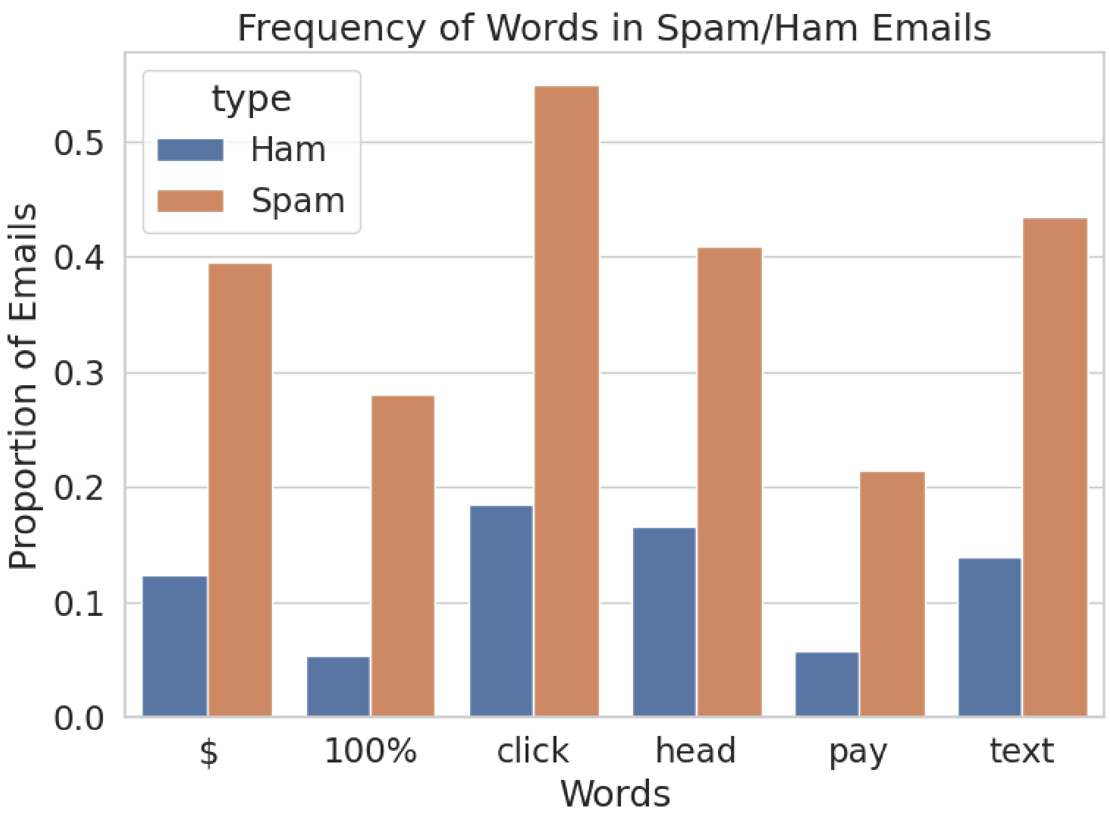
Create your bar chart in the following cell:

[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.
plt.figure(figsize=(8,6))

chosen_words = ['head', 'click', 'pay', 'text', '100%', '$']
word_indicators = words_in_texts(chosen_words, train['email'])
word_indicators_df = pd.DataFrame(word_indicators, columns=chosen_words)
word_indicators_df['type'] = train['spam'].replace({1: 'Spam', 0: 'Ham'})
melted_df = word_indicators_df.melt(id_vars='type')
proportion_df = melted_df.groupby(['type', 'variable']).agg(np.mean).
↪reset_index()

sns.barplot(data=proportion_df, x='variable', y='value', hue='type')
plt.title('Frequency of Words in Spam/Ham Emails')
plt.xlabel('Words')
plt.ylabel('Proportion of Emails')

plt.tight_layout()
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.

6 Part 4: 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!

6.1 Question 4

We've given you 5 words that might be useful as features to distinguish spam/ham emails. Use these words and the `train` `DataFrame` to create two `NumPy` arrays: `X_train` and `Y_train`. `X_train` should be a 2D array of 0s and 1s created 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 design matrix ($\mathbb{X}$) are correct and that your features and labels are binary (i.e., consist only of 0s and 1s). It does not check that your function is correct; that was verified in Question 2.

[14]:
```
some_words = ['drug', 'bank', 'prescription', 'memo', 'private']

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

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

[14]:
```
(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]))
```

6.2 Question 5

Now that we have matrices, we can build a model with `sklearn`! 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 0.76.

The provided tests check that you initialized your logistic regression model correctly.

[16]:
```
from sklearn.linear_model import LogisticRegression

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

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

```
Training Accuracy:  0.7576201251164648
```

[17]:
```
assert np.allclose(my_model.coef_, np.array([[ 0.3876794 ,  1.41303343,  2.
 ↪04437707, -0.53676679,  0.92334944]]))
```

7 Part 5: 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 the accuracy of the model on the training set, which may be a misleading 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 make use of the data we held out for model validation and comparison.

Presumably, our classifier will be used for **filtering**, or 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 of emails flagged as spam that are actually spam. Mathematically, $\frac{\text{TP}}{\text{TP+FP}}$.

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

False positive rate: Measures the proportion of ham emails that were incorrectly flagged as spam. Mathematically, $\frac{\text{FP}}{\text{FP+TN}}$.

One quick mnemonic to remember the formulas is that **P**recision involves T**P** and F**P**, Recall does not. In the final, the reference sheet will also contain the formulas shown above, but you should be able to interpret what they mean and their importance depending on the context.

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.

7.1 Question 6a

Suppose we have a hypothetical classifier called the "zero predictor." For any inputted email, the zero predictor *always* predicts 0 (it never makes a prediction of 1 for any email). 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`? Assign `zero_predictor_fp` to the number of false positives and `zero_predictor_fn` to the number of false negatives for the hypothetical zero predictor on the training data.

The public tests only check that you have assigned appropriate types of values to each response variable but do not check that your answers are correct. That is, we only check that the number of false positives and false negatives should be greater than or equal to 0.

```
[19]: zero_predictor_fp = 0
      zero_predictor_fn = np.sum(Y_train == 1)
      zero_predictor_fp, zero_predictor_fn
```

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

7.2 Question 6b

What is the accuracy and recall of the zero predictor on the training data? Do not use any `sklearn` functions to compute these performance metrics.

The public tests only check that you have assigned appropriate types of values to each response variable but do not check that your answers are correct. That is, we only check that proportions or *percentages (like precision, recall, accuracy) lie in the interval [0, 1].*

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

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

7.3 Question 6c

Explain your results in `q6a` and `q6b`. How did you know what to assign to `zero_predictor_fp`, `zero_predictor_fn`, `zero_predictor_acc`, and `zero_predictor_recall`?

zero_predictor_fp: The zero predictor never predicts 1, i.e., it never predicts postive, no matter true or false. So false postive is always 0.

zero_predictor_fn: False negative of the zero predictor is the count of spam emails in Y_train.

zero_predictor_acc: The accuracy is correct predictions (true negative) / total predictions.

zero_predictor_recall: Since the zero predictor never predicts spam, True Positives = 0. So recall = TP/(TP+FN) = 0.

7.4 Question 6d

Compute the precision, recall, and false positive rate of the `LogisticRegression` classifier `my_model` from Question 5. Do **not** use any `sklearn` functions to compute performance metrics; the only `sklearn` method you may use here is `.predict` to generate model predictions using `my_model` and `X_train`.

The public tests only check that you have assigned appropriate types of values to each response variable but do not check that your answers are correct. That is, we only check that proportions or percentages (like precision, recall, false positive rate) lie in the interval [0, 1].

```
[23]: Y_train_hat = my_model.predict(X_train)

      TP = np.sum((Y_train_hat == 1) & (Y_train == 1))
      TN = np.sum((Y_train_hat == 0) & (Y_train == 0))
      FP = np.sum((Y_train_hat == 1) & (Y_train == 0))
      FN = np.sum((Y_train_hat == 0) & (Y_train == 1))
      logistic_predictor_precision = TP / (TP + FP)
      logistic_predictor_recall = TP / (TP + FN)
      logistic_predictor_fpr = FP / (FP + TN)

      print(f"{TP=}, {TN=}, {FP=}, {FN=}")
      print(f"{logistic_predictor_precision=:.2f}, {logistic_predictor_recall=:.2f},␣
       ↪{logistic_predictor_fpr=:.2f}")
```

```
TP=219, TN=5473, FP=122, FN=1699
logistic_predictor_precision=0.64, logistic_predictor_recall=0.11,
logistic_predictor_fpr=0.02
```

7.5 Question 6e

Is the number of false positives produced by the logistic regression classifier `my_model` strictly greater than the number of false negatives produced? Assign to `q6e` an expression that evaluates to give your answer (`True` or `False`).

[25]:
```
q6e = False
q6e
```

[25]:
```
False
```

7.6 Question 6f

How does the accuracy of the logistic regression classifier `my_model` compare to the accuracy of the zero predictor?

Accuracy of my_model: (219+5473)/(7513) = 0.7576201251164648, which is identical to the accuracy of the zero predictor.

7.7 Question 6g

Given the word features provided in Question 4, discuss why the logistic regression classifier `my_model` may be performing poorly.

Hint: Think about how prevalent these words are in the email set.

The chosen word features (‘drug’, ‘bank’, ‘prescription’, ‘memo’, ‘private’) may not help distinguishing spam from ham very well. These words are either too rare or too frequently in both spam and ham emails.

7.8 Question 6h

Would you prefer to use the logistic regression classifier `my_model` or the zero predictor classifier for a spam filter? Why? Describe your reasoning and relate it to at least one of the evaluation metrics you have computed so far.

I prefer to use the logistic regression classifier my_model. The zero predictor never predicts any email as spam since it always predicts 0. It completely fails to identify any spam emails since the recall is 0. This is unacceptable for a spam filter since it can’t filter any spam emails.