proj2a.ipynb Download

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

Project 2A: Spam/Ham Classification

Feature Engineering, Logistic Regression

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*

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.

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.

In [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.
```

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

Part 1: Initial Analysis

In [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)
```

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.

In [4]:
```
import zipfile
with zipfile.ZipFile('spam_ham_data.zip') as item:
    item.extractall()
```

In [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()
```

Out [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

In [6]:
```
original_training_data[original_training_data['spam']==1][70:80]
```

Out [6]:

	id	subject
308	308	Subject: Is Neotropin right for you?\n
311	311	Subject: Brighten Those Teeth\n
312	312	Subject: The database that Bill Gates doesnt w...
322	322	Subject: Keep porn free...\n
324	324	Subject: Real Drugs-Viagra and Phentrimine!\n
326	326	Subject: RE: MEN & WOMEN, TURBO BOOST YOUR DRI...
327	327	Subject: Financial Freedom That You Deserve...\n

```
330  330            Subject: VTGE   7/22/2002 10:24:24 PM\n
337  337  Subject: Low Cost High Rated Insurance. Why Pa...
338  338  Subject: Have You Dreamed Of Your Own Home Bas...
```

	email	spam
308	<html><body> <a href=http://www.vitafac...	1
311	<!-- saved from url=3d(0022)http://internet.e-...	1
312	important notice: regarding your domain name\...	1
322	this is a multi-part message in mime format.\n...	1
324	<html>\n <head>\n <meta http-equiv="content-ty...	1
326	<html>\n \n <head>\n <meta http-equiv=3d"conte...	1
327	<html><head><title></title></head>\n <body bgc...	1
330	dear me ,\n \n <!-- saved from url=(0022)http:...	1
337	<html>\n <head>\n <body>\n \n <table bgcolor=3...	1
338	------=_nextpart_000_00x9_70a11c1d.e1232j43\n ...	1

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.

In [7]:
```
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
```

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.

In [8]:
```
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.

- I noticed that the spam email tends to have html and body tags and is much longer than the ham email. It may because the spam email would contains the advertisements or others, it make us easy to observe which emails tends to be spam.

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.**

In [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)
```

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.

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.

In [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.
    '''
    rt_lists = []
    for i in texts:
        subset_ls = []
        for j in words:
            if j in i:
                subset_ls.append(1)
            else:
                subset_ls.append(0)
        rt_lists.append(subset_ls)
    indicator_array = np.asarray(rt_lists)
    return indicator_array
```

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

Out [11]:
```
q2 results: All test cases passed!
```

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

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

In [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

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`.

In [13]:

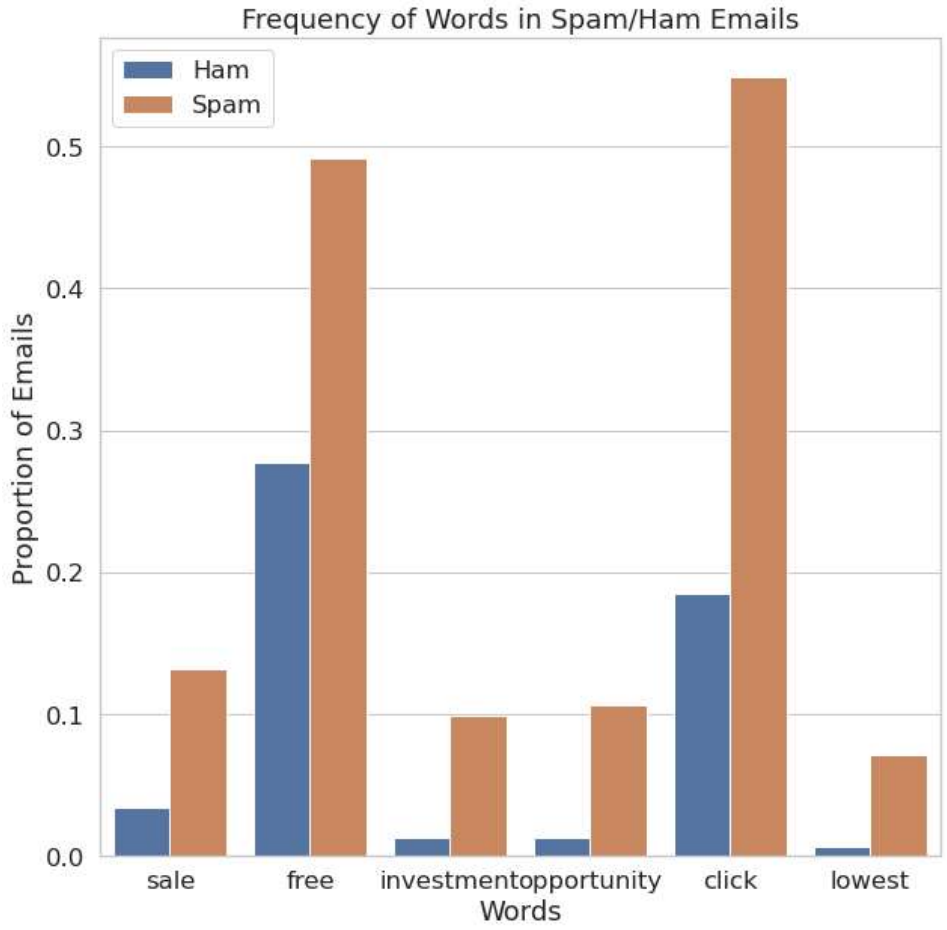
```
train = train.reset_index(drop=True) # We must do this in order to preserve
the ordering of emails to labels for words_in_texts

word_freq_list = words_in_texts(['sale', 'free','investment','opportunity',
'click', 'lowest'], train['email'])

sale = [item[0] for item in word_freq_list]
free = [item[1] for item in word_freq_list]
investment = [item[2] for item in word_freq_list]
opportunity = [item[3] for item in word_freq_list]
click = [item[4] for item in word_freq_list]
lowest = [item[5] for item in word_freq_list]

train_subset = pd.DataFrame(data= {'sale':sale, 'free':free,
'investment':investment, 'opportunity':opportunity, 'click':click,
                                   'lowest': lowest,
'spam_or_not':train['spam']})
new_col = train_subset['spam_or_not'].replace({0:'Ham',1:'Spam'}, inplace =
True)
melted_train = train_subset.melt('spam_or_not')

plt.figure(figsize = (10,10))
sns.barplot(x="variable", y="value", hue = 'spam_or_not', ci =None, data =
melted_train)
plt.xlabel("Words")
plt.ylabel("Proportion of Emails")
plt.title('Frequency of Words in Spam/Ham Emails')
plt.gca().legend().set_title('')
```



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.

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!

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.

In [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]
```

Out [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]))
```

In [15]:
```
grader.check("q4")
```

Out [15]:
```
q4 results: All test cases passed!
```

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.

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

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

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

```
Training Accuracy:  0.7576201251164648
```

In [17]:
```
grader.check("q5")
```

Out [17]:
```
q5 results: All test cases passed!
```

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:  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 6

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.

In [18]:
```
zero_predictor_fp = 0
zero_predictor_fn = sum(1 == Y_train)
zero_predictor_fp, zero_predictor_fn
```

Out [18]:
```
(0, 1918)
```

In [19]:
```
grader.check("q6a")
```

Out [19]:
```
q6a results: All test cases passed!
```

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.

In [20]:
```
zero_predictor_acc = sum(0 == Y_train)/len(Y_train)
zero_predictor_recall = 0 / (0 + zero_predictor_fn)
zero_predictor_acc, zero_predictor_recall
```

Out [20]:
```
(0.7447091707706642, 0.0)
```

In [21]:
```
grader.check("q6b")
```

Out [21]:
```
q6b results: All test cases passed!
```

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.

- 6a: Since the classifier zero_predictor always predicts 0, hence false positive is 0. Also the zero_predictor will never predict the true positive, hence the false negative is the total number of spams.
- 6b: the accuracy is the number of hams divided by the total number of emails.It tells us how accurate our classifier is.
- FP:the ham emails gets flagged as spam and filtered out of the inbox. Since we are predict all 0 in this case, there is no false positive in this case.
- FN: the spam emails gets mislabeled as ham and ends up in the inbox. Since we labelled all the emails as zero, so all the emails that Y_train = 1 is the false negative.
- accuracy: the accuracy is the number of hams divided by the total number of emails. Since we are predict all 0 in this case, the sum of true positive plus true negative is equal to the sum of Y_train == 0.
- recall: Since true positive is 0 in our case, the result is 0.

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.

In [32]:
```
Y_pred = model.predict(X_train)
tp = sum((Y_pred == Y_train) & (Y_train == 1))
fp = sum((Y_pred != Y_train) & (Y_train == 0))
fn = sum((Y_pred != Y_train) & (Y_train == 1))
tn = sum((Y_pred == Y_train) & (Y_train == 0))

logistic_predictor_precision = tp/(tp + fp)
logistic_predictor_recall = tp / (tp + fn)
logistic_predictor_far = fp / (fp +tn)
```

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

Out [24]:
```
q6d results: All test cases passed!
```

Question 6e

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

- Since the fp=122 and the fn=1699, there is more false negatives than false positive.

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 logistic regression classifier is better since the predict_0 classifier yields 74.4% which is less than 75.76%
2. As we can see, the encoded feature in X_trains have many rows with all 0. It means the words we are choose barely occur in the training set, this makes us get many redundant data and the classifer can't use them to distinguish spam or ham.
3. I would choose the zero_predictor. As we can see, the logistic regression classifier has 2% False-alarm rate, while the zero_predictor has the 0% false-alarm rate, I'd rather filter less spam emails but filter out my important main email.

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.

In [30]:
```
grader.check_all()
```

Out [30]:
```
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!
```

Submission

proj2b.ipynb Download

In [256]:
```
# Initialize Otter
import otter
grader = otter.Notebook("proj2b.ipynb")
```

Project 2B: Spam/Ham Classification - Build Your Own Model

Feature Engineering, Classification, Cross Validation

Due Date: Sunday 4/28, 11:59 PM PDT

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 be building and improving on the concepts and functions that you implemented in Project 2A to create your own classifier to distinguish spam emails from ham (non-spam) emails. We will evaluate your work based on your model's accuracy and your written responses in this notebook.

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

- 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

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 project so that you are made aware.

In [257]:
```
# Run this cell to suppress all FutureWarnings
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

Score Breakdown

Question	Points
1	6
2a	4

Question	Points
2b	2
3	3
4	15
Total	30

Setup and Recap

Here we will provide a summary of Project 2A to remind you of how we cleaned the data, explored it, and implemented methods that are going to be useful for building your own model.

In [258]:
```
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 and Cleaning Data

Remember that 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 cell 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.

In [259]:
```
import zipfile
with zipfile.ZipFile('spam_ham_data.zip') as item:
    item.extractall()
```

In [260]:
```
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()
```

Out [260]:

	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

Feel free to explore the dataset above along with any specific spam and ham emails that interest you. Keep in mind that our data may contain missing values, which are handled in the following cell.

In [261]:
```
# Fill any missing or NAN values
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
```

Training/Validation Split

Recall that 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 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.

As in Project 2A, we set the seed (random_state) to 42. **Do not modify this in the following questions, as our tests depend on this random seed.**

In [262]:
```
# 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)

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

Feature Engineering

In order to train a logistic regression model, we need a numeric feature matrix X and a vector of corresponding binary labels y. To address this, in Project 2A, we implemented the function `words_in_texts`, which creates numeric features derived from the email text and uses those features for logistic regression.

For this project, we have provided you with an implemented version of `words_in_texts`. Remember that the function outputs 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.

In [263]:
```
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.
    '''
    import numpy as np
    indicator_array = 1 * np.array([texts.str.contains(word) for word
in words]).T
    return indicator_array
```

Run the following cell to see how the function works on some dummy text.

In [264]:
```
words_in_texts(['hello', 'bye', 'world'], pd.Series(['hello', 'hello
worldhello']))
```

Out [264]:
```
array([[1, 0, 0],
       [1, 0, 1]])
```

EDA and Basic Classification

In Project 2A, we proceeded to visualize the frequency of different words for both spam and ham emails, and used `words_in_texts(words, train['email'])` to directly to train a classifier. We also provided a simple set of 5 words that might be useful as features to distinguish spam/ham emails.

We then built a model using the using the `LogisticRegression` classifier from `scikit-learn`.

Run the following cell to see the performance of a simple model using these words and the `train` dataframe.

In [265]:
```
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]
```

Out [265]:
```
(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 [266]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver = 'lbfgs')
model.fit(X_train, Y_train)

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

```
Training Accuracy:  0.7576201251164648
```

Evaluating Classifiers

In our models, we are evaluating accuracy on the training set, which may provide a misleading accuracy measure. In Project 2A, we calculated various metrics to lead us to consider more ways of evaluating a classifier, in addition to overall accuracy. Below is a reference to those concepts.

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

False-alarm rate measures the proportion $\frac{\text{FP}}{\text{FP}+\text{TN}}$ 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.

Moving Forward - Building Your Own Model

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. You will be able to see your accuracy on the entire test set when submitting to Gradescope.

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 decision trees, 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 1, 2, and 3 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. Alternatively, you can perform cross-validation on the entire training set.

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

word_freq_list = words_in_texts(['refund', 'success','honor','call',
'earn', 'extra'], train_copy['email'])

sale = [item[0] for item in word_freq_list]
free = [item[1] for item in word_freq_list]
investment = [item[2] for item in word_freq_list]
opportunity = [item[3] for item in word_freq_list]
click = [item[4] for item in word_freq_list]
lowest = [item[5] for item in word_freq_list]

train_subset = pd.DataFrame(data= {'sale':sale, 'free':free,
'investment':investment, 'opportunity':opportunity, 'click':click,
                                   'lowest': lowest,
'spam_or_not':train['spam']})
new_col = train_subset['spam_or_not'].replace({0:'Ham',1:'Spam'},
inplace = True)
melted_train = train_subset.melt('spam_or_not')

plt.figure(figsize = (10,10))
sns.barplot(x="variable", y="value", hue = 'spam_or_not', ci =None,
data = melted_train)
plt.xlabel("Words")
plt.ylabel("Proportion of Emails")
plt.title('Frequency of Words in Spam/Ham Emails')
plt.gca().legend().set_title('')
```

In [ ]:
```
feature_words = ["<html>", "<head>", "<body>", "<br>", "font","
<title>",
                 "div>", "<center>", "align", "table", "offer", "deal",
                  "please", "address", "money","!!", "100%", "sale",
"investment",
                 "business", "guarantee","credit", "earn",
                  "did", "free", "today", "information", "share",
                 "url:", "click", "height", "fill out", "unsubscribe",
"shipping"]

X_train = words_in_texts(feature_words, train.loc[:,'email'])
Y_train = train.loc[:,'spam']

X_Val = words_in_texts(feature_words, val['email'])
Y_Val = val['spam'].values

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

val_accuracy = model.score(X_Val, Y_Val)
train_accuracy = model.score(X_train, Y_train)
print('Validation Accuracy:', val_accuracy)
print('train Accuracy:', train_accuracy)
```

Question 1: 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. First I utilize the "Frequency of Words in Spam/Ham Emails" plot we generate in proj-2a. I repeatly apply the words that I come up with and selected the words which has a significant difference between spam and ham.
2. When I was thinking of the words to try, I first select the spam's row's out and observe that it contains many html related format, hence I selected the frequent words. Which didn't work is sometimes the words that I choose has a low occurrence in both spam and ham. For example "\ ", Then I slightly change its format to "div>" and it worked.
3. Some words just didn't appears as I expect. For example, at the beginning I thought "Congratulations" or "Congratz" should show up greatly in both case but turns out it didn't. And I tried the word "unsubscribe" randomly but turns out it appeared frequently in spam email. It just a bit changed my mind that the spam emails are not that malicious.

Optional: Build a Decision Tree model with reasonably good accuracy. What features does the decision tree use first?

Question 2: 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 in Project 2A. 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.

Question 2a

Generate your visualization in the cell below.

In [ ]:
```
corr_df = train
words = ["money","offer", "final","sale","price"]

corr_df['money'] = words_in_texts(['money'],train['email']) #each
list tells you whether word is in the ith email or not
corr_df['offer'] = words_in_texts(['offer'],train['email'])
corr_df['final'] = words_in_texts(['final'],train['email'])
corr_df['sale'] = words_in_texts(['sale'],train['email'])
corr_df['price'] = words_in_texts(['price'],train['email'])
correlation = corr_df[words]
sns.heatmap(correlation.corr());
plt.title("heat map of words pairs");
```

Question 2b

Write your commentary in the cell below.

I choose to generate a heatmap graph of 5 words pairs which I feel they have the high probability appear together hence may have a high correlation. Since the high correlated words just restate the information and even worse, multicollinearity sometimes will mess up the prediction so I want to prevent this situation. As the graph above shows, most pairs don't have high correlation, which is good, but as for the offer and price pair, the correlation is about 0.4, hence I evict the word "price" anyway.

Question 3: 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 20 to see how to plot an ROC curve.

Hint: You'll want to use the `.predict_proba` method for your classifier instead of `.predict` so you get probabilities instead of binary predictions.

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

fpr, tpr, _ = roc_curve(Y_train, [i[1] for i in
model.predict_proba(X_train)])
plt.step(fpr, tpr)
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.ylim([-0.01, 1.01])
```

```
plt.xlim([-0.01, 1.01])
plt.title("ROC curve");
```

Question 4: 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 "k 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.

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

In [ ]:
```
X_test = words_in_texts(feature_words, test['email'])
test_predictions = model.predict(X_test)
```

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

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

Note that the file will appear in your DataHub, you must navigate to the `hw11` directory in your DataHub to download the file.

In [ ]:
```
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.')
```

Congratulations! You have completed Project 2B!

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

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

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!**

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

▸ **proj2b.pdf** Download

▸ **.OTTER_LOG** Download

▸ **__zip_filename__** Download

Project 2B

● Graded

Select each question to review feedback and grading details.

Student
Luna Tian

Total Points
3 / 3 pts

Autograder Score
3.0 / 3.0

Passed Tests
Public Tests
q4 (3/3)