In [1]:
```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('proj2.ok')
```

```
=====================================================================
Assignment: proj2
OK, version v1.13.11
=====================================================================
```

Project 2: Spam/Ham Classification

Feature Engineering, Logistic Regression, Cross Validation

Due Date: Sunday 11/24/19, 11:59PM

Collaboration Policy

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

Collaborators: *list collaborators here*

This Assignment

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

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

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

Warning

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

Score Breakdown

Question	Points
1a	1
1b	1
1c	2
2	3
3a	2
3b	2
4	2
5	2

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

Part I - Initial Analysis

In [2]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

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

Loading in the Data

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

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

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

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

1. `id` : An identifier for the training example
2. `subject` : The subject of the email
3. `email` : The text of the email
4. `spam` : 1 if the email is spam, 0 if the email is ham (not spam)

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

In [3]:
```
from utils import fetch_and_cache_gdrive
fetch_and_cache_gdrive('1SCASpLZFKCp2zek-toR3xeKX3DZnBSyp', 'train.csv')
fetch_and_cache_gdrive('1ZDFo9OTF96B5GP2Nzn8P8-AL7CTQXmC0', 'test.csv')

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

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

original_training_data.head()
```

```
Using version already downloaded: Sat Nov 23 12:11:21 2019
MD5 hash of file: 0380c4cf72746622947b9ca5db9b8be8
Using version already downloaded: Sat Nov 23 12:11:22 2019
MD5 hash of file: a2e7abd8c7d9abf6e6fafc1d1f9ee6bf
```

Out[3]:

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

Question 1a

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

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

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

In [4]:
```
for i in np.arange(4):
    print('The number of NaN values in column '+str(i+1)+' is: '+str(original_training_data.iloc[:,
    if original_training_data.iloc[:,i].isnull().sum().sum()!=0:
        original_training_data.iloc[:,i].fillna('',inplace=True)
```

```
The number of NaN values in column 1 is: 0
The number of NaN values in column 2 is: 6
The number of NaN values in column 3 is: 0
The number of NaN values in column 4 is: 0
```

In [5]:
```
ok.grade("q1a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooook] 100.0% passed
```

Question 1b

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

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

In [6]:
```
first_ham = original_training_data.query('spam==0')['email'].iloc[0]
first_spam = original_training_data.query('spam==1')['email'].iloc[0]
print(first_ham)
print(first_spam)
```

```
url: http://boingboing.net/#85534171 (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/ (http://www.aldaily.com/)
 [2] http://www.quicktopic.com/boing/h/zlfterjnd6jf (http://www.quicktopic.com/boing/h/zlfterjnd6j
f)



<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>
```

In [7]:
```
ok.grade("q1b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooook] 100.0% passed
```

Question 1c

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

The spam email started with an attention getter whose tone is more interrogative, which makes the reader become inquisitive and fall in the trap. While the ham email is in a more declarative tone. Despite of tone, we can also detect the difference by wordings. For example, the spam email has words such as "guaranteed". While in the ham email, we don't see this kind of strongly affirmative words.

Training Validation Split

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

In [8]:
```
from sklearn.model_selection import train_test_split

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

Basic Feature Engineering

We would like to take the text of an email and predict whether the email is ham or spam. This is a *classification* problem, so we can use logistic regression to train a classifier. Recall that to train an logistic regression model we need a numeric feature matrix 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 when you are trying to increase your accuracy.

Question 2

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

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

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

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

In [9]:
```
def words_in_texts(words, texts):
    '''
    Args:
        words (list-like): words to find
        texts (Series): strings to search in

    Returns:
        NumPy array of 0s and 1s with shape (n, p) where n is the
        number of texts and p is the number of words.
    '''
    indicator_array=[]
    for i in np.arange(len(texts)):
        for j in np.arange(len(words)):
            if words[j] in texts[i]:
                indicator_array.append(1)
            else:
                indicator_array.append(0)

    indicator_array = np.split(np.array(indicator_array),len(texts))
    return np.array(indicator_array)
```

In [10]:
```
words_in_texts(['hello', 'bye', 'world'],
                   pd.Series(['hello', 'hello worldhello']))
```

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

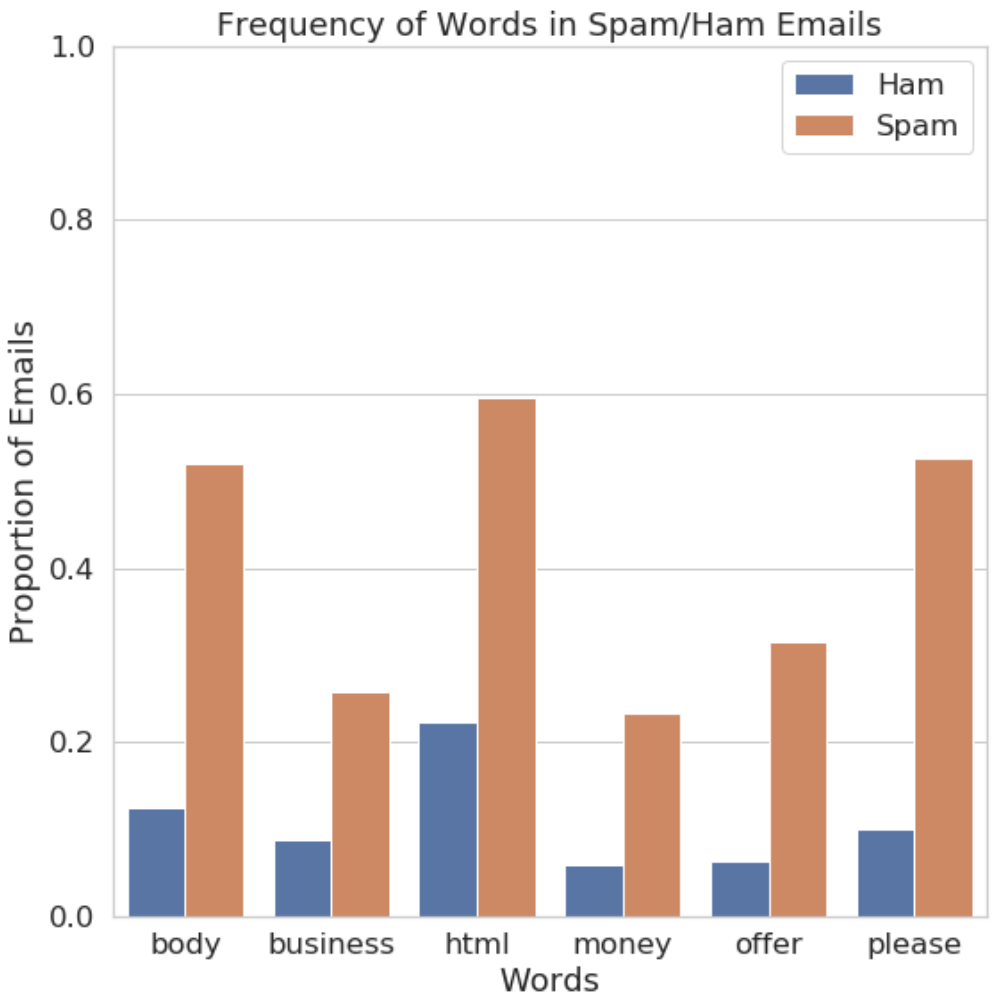
In [11]:
```
ok.grade("q2");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooook] 100.0% 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.



Hint:

- 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 some words column and a type column. You can think o
display(df);
display(Markdown("> `melt` will turn columns into variable, notice how `word_1` and `word_2` become
display(df.melt("type"))
```

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

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

> `melt` will turn columns into variable, notice how `word_1` and `word_2` become `variable`, their values are stored in the value column

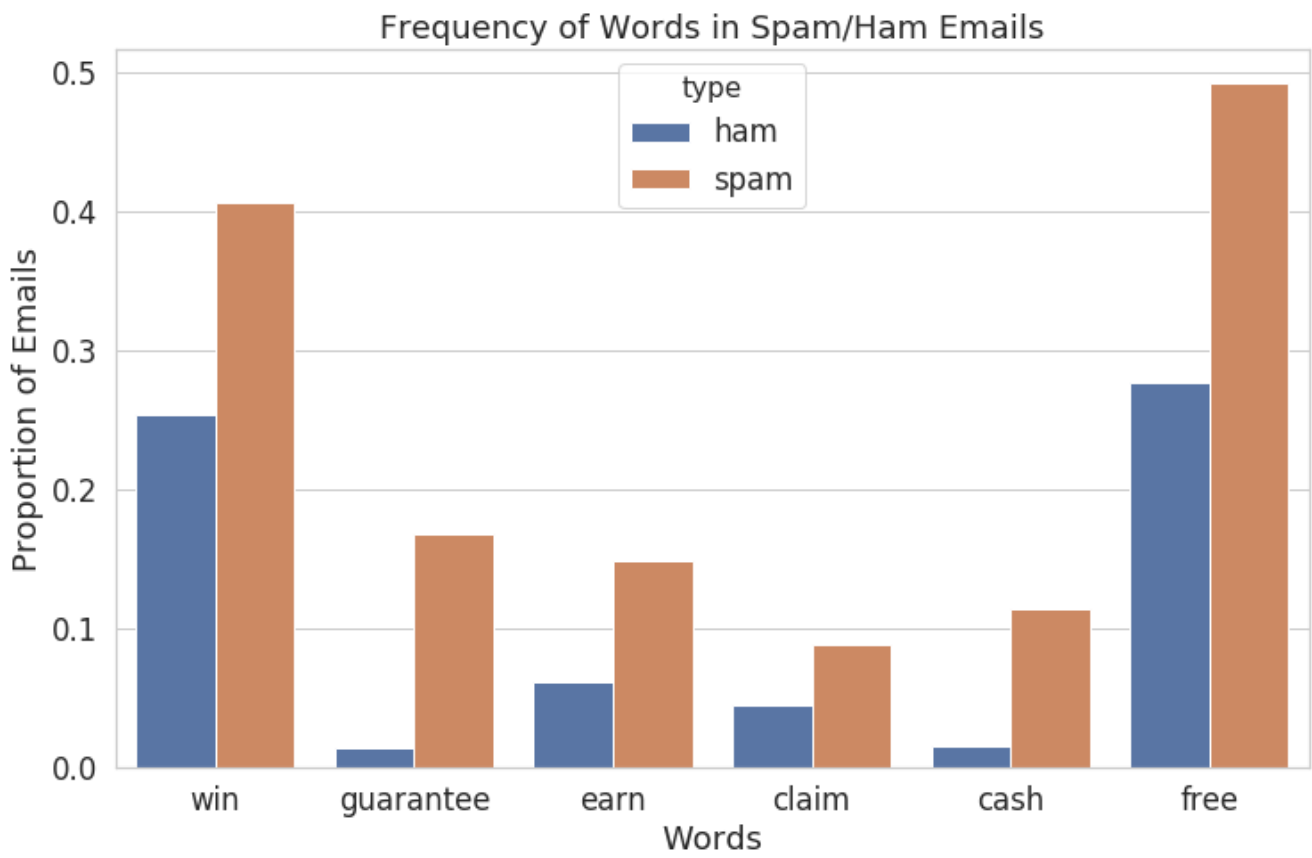
	type	variable	value
0	spam	word_1	1
1	ham	word_1	0
2	ham	word_1	1
3	ham	word_1	0
4	spam	word_2	0
5	ham	word_2	1
6	ham	word_2	0
7	ham	word_2	1

Question 3a

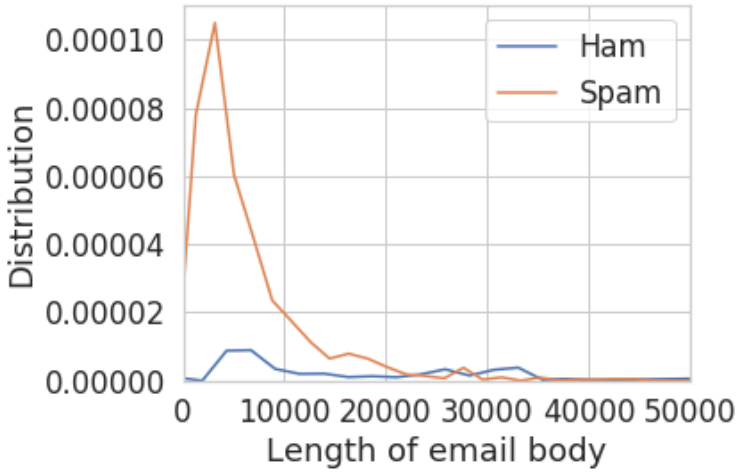
Create a bar chart like the one above comparing the proportion of spam and ham emails containing certain words. Choose a set of words that are different from the ones above, but also have different proportions for the two classes. Make sure to only consider emails from `train`.

In [13]:

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

words=['win','guarantee','earn','claim','cash','free']

wordsintexts=words_in_texts(words,train['email'])

win=wordsintexts[:,0]
guarantee=wordsintexts[:,1]
earn=wordsintexts[:,2]
claim=wordsintexts[:,3]
cash=wordsintexts[:,4]
free=wordsintexts[:,5]

ham_spam=train['spam'].replace(0,'ham').replace(1,'spam')

ice=pd.DataFrame({'win':win,
                  'guarantee':guarantee,
                  'earn':earn,
                  'claim':claim,
                  'cash':cash,
                  'free':free,
                  'type':ham_spam})

water=ice.melt('type')

plt.figure(figsize=(12,7.417))
sns.barplot(x = 'variable', y = 'value', hue='type', data=water,ci=None)
plt.title('Frequency of Words in Spam/Ham Emails')
plt.xlabel('Words')
plt.ylabel('Proportion of Emails');
```



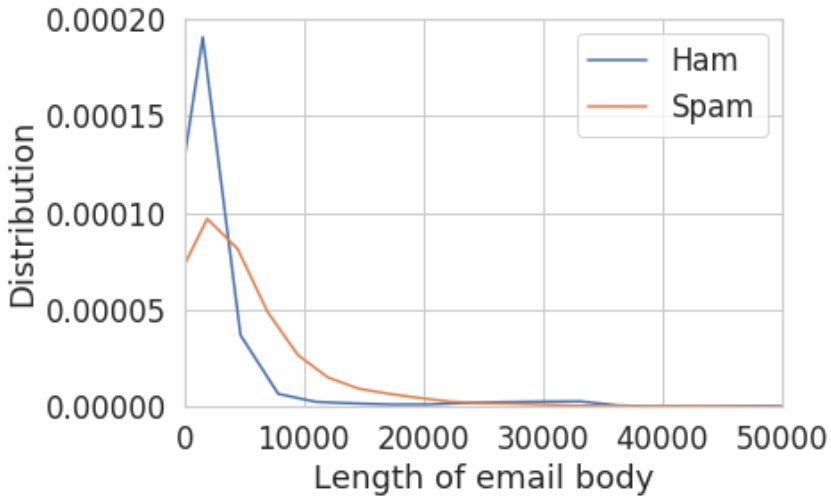
When the feature is binary, it makes sense to compare its proportions across classes (as in the previous question). Otherwise, if the feature can take on numeric values, we can compare the distributions of these values for different classes.



Question 3b

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

In [14]:
```
sns.distplot(train.query("spam==0")['email'].str.len(),label='Ham',hist=False)
sns.distplot(train.query("spam==1")['email'].str.len(),label='Spam',hist=False)

plt.legend()
plt.xlim(0,50000)
plt.xlabel('Length of email body')
plt.ylabel('Distribution');
```



Basic Classification

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

Question 4

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

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

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

The provided tests check that the dimensions of your feature matrix (X) are correct, and that your features and labels are binary (i.e. consists of 0 and 1, no other values). It does not check that your function is correct; that was verified in a previous question.

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

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

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

Out[15]:
```
(array([[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0]]), 0    0
 1    0
 2    0
 3    0
 4    0
 Name: spam, dtype: int64)
```

In [16]:
```
ok.grade("q4");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooook] 100.0% passed
```

Question 5

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

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

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

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

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

```
Training Accuracy:  0.7576201251164648
```

In [18]:
```
ok.grade("q5");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 1
    Failed: 0
[oooooooook] 100.0% passed
```

Evaluating Classifiers

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

Presumably, our classifier will be used for **filtering**, i.e. preventing messages labeled `spam` from reaching someone's inbox. There are two kinds of errors we can make:

- False positive (FP): a ham email gets flagged as spam and filtered out of the inbox.
- False negative (FN): a spam email gets mislabeled as ham and ends up in the inbox.

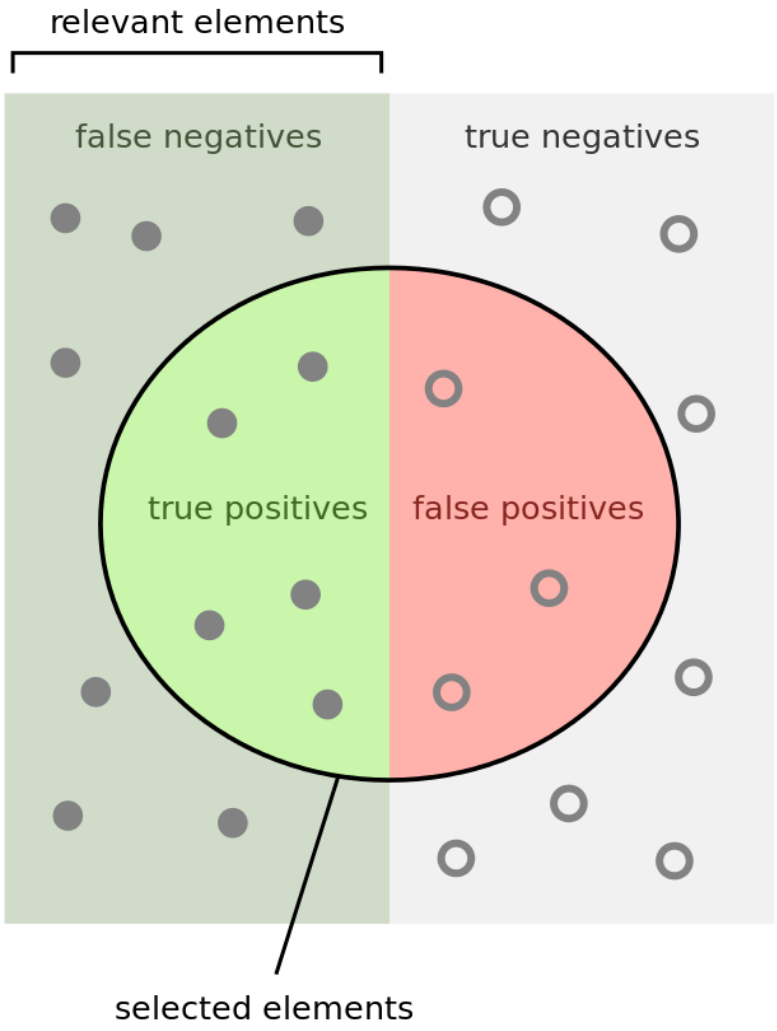
These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

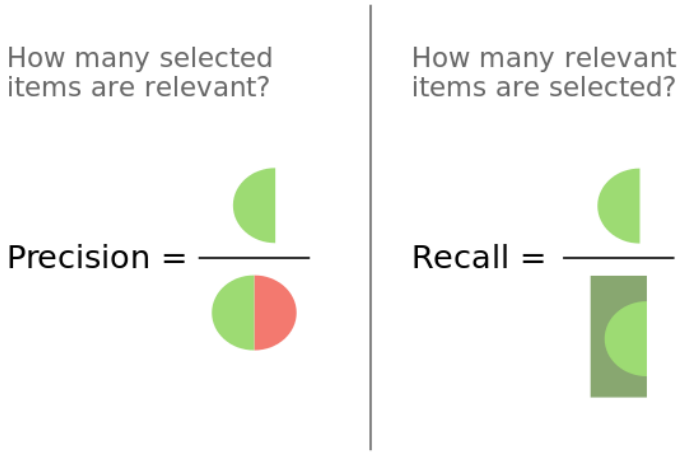
Precision measures the proportion $\frac{\text{TP}}{\text{TP+FP}}$ of emails flagged as spam that are actually spam.

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 following image might help:





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

Question 6a

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

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

In [19]:
```
zero_predictor_fp = 0
zero_predictor_fn = len(train[train['spam']==1])
zero_predictor_fn
```

Out[19]: 1918

In [20]:
```
ok.grade("q6a");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 6b

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

In [21]:
```
zero_predictor_acc = len(train[train['spam']==0])/len(X_train)
zero_predictor_recall = 0
zero_predictor_acc
```

Out[21]: 0.7447091707706642

In [22]:
```
ok.grade("q6b");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 6c

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

For 6(a),the classifier zero_predictor will predict 0 all the time. False positive is 0 because we don't predict any email to be a spam email. False negative value represents that there are 1918 spam emails marked as ham email incorrectly. For 6b), the 74% accuracy of zero predictor accuracy represents the proportion of amount of ham emails out of total emails. Recall is 0 because of we predicted 0 all the time and the true positive is always 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.

In [23]:
```
logistic_predictor_precision = sum(model.predict(X_train)&Y_train)/(sum(model.predict(X_train)&Y_tr
logistic_predictor_recall = sum(model.predict(X_train)&Y_train)/(sum(model.predict(X_train)&Y_train
logistic_predictor_far = sum(model.predict(X_train)&(~Y_train))/(sum(model.predict(X_train)&(~Y_tra
```

In [24]:
```
ok.grade("q6d");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

Question 6e

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

From 6d, we get the precision of 64.22% and recall of 11.41%. Since the precision > recall, FN > FP. Therefore, there are more false negative than false positive when using the logistic regression classifier from Question 5.

Question 6f

1. Our logistic regression classifier got 75.6% prediction accuracy (number of correct predictions / total). How does this compare with predicting 0 for every email?
2. Given the word features we gave you above, name one reason this classifier is performing poorly. Hint: Think about how prevalent these words are in the email set.
3. Which of these two classifiers would you prefer for a spam filter and why? Describe your reasoning and relate it to at least one of the evaluation metrics you have computed so far.

1)The accuracy of the 0 predicting model is only 74.4709%, which is less than the accuracy (75.6%) of the logistic regression classifier.

2) The word choices might be very generic and prevalent in both the spam and ham emails.

3) The logistic regression model is better because it has a higher prediction accuracy. To make improvements on the prediction accuracy, I will do some feature engineering, look for non-prevalent words, and change the word choices.

Part II - Moving Forward

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

Kaggle limits you to four submissions per day. This means you should start early so you have time if needed to refine your model. You will be able to see your accuracy on the entire set when submitting to Kaggle (the accuracy that will determine your score for question 10).

Here are some ideas for improving your model:

1. Finding better features based on the email text. Some example features are:
    A. Number of characters in the subject / body
    B. Number of words in the subject / body

C. Use of punctuation (e.g., how many '!' were there?)
D. Number / percentage of capital letters
E. Whether the email is a reply to an earlier email or a forwarded email

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

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

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

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

Step 1 Clean the Data

We first need to remove all the HTML tags, and new line symbol \n. We need to extract out the text from the HTML so that we can find better words and get a better and cleaner result

In [25]:
```
#Remove all HTML related tags and newline \n
train['email']=train['email'].str.replace('<[^<]+?>','').str.replace('\n','')
train
```

Out[25]:

	id	subject	email	spam
0	7657	Subject: Patch to enable/disable log\n	while i was playing with the past issues, it a...	0
1	6911	Subject: When an engineer flaps his wings\n	url: http://diveintomark.org/archives/2002/10/...	0
2	6074	Subject: Re: [Razor-users] razor plugins for m...	no, please post a link! fox ----- original me...	0
3	4376	Subject: NYTimes.com Article: Stop Those Press...	this article from nytimes.com has been sent t...	0
4	5766	Subject: What's facing FBI's new CIO? (Tech Up...	tech update today ...	0
...	...	...	...	...
7508	5734	Subject: [Spambayes] understanding high false ...	>>>>> "tp" == tim peters writes: >> first ...	0
7509	5191	Subject: Reach millions on the internet!!\n	dear consumers, increase your business sales!...	1
7510	5390	Subject: Facts about sex.\n	forwarded-by: flower did you know that you c...	0
7511	860	Subject: Re: Zoot apt/openssh & new DVD playin...	on tue, oct 08, 2002 at 04:36:13pm +0200, matt...	0
7512	7270	Subject: Re: Internet radio - example from a c...	chris haun wrote: > > we would need someone t...	0

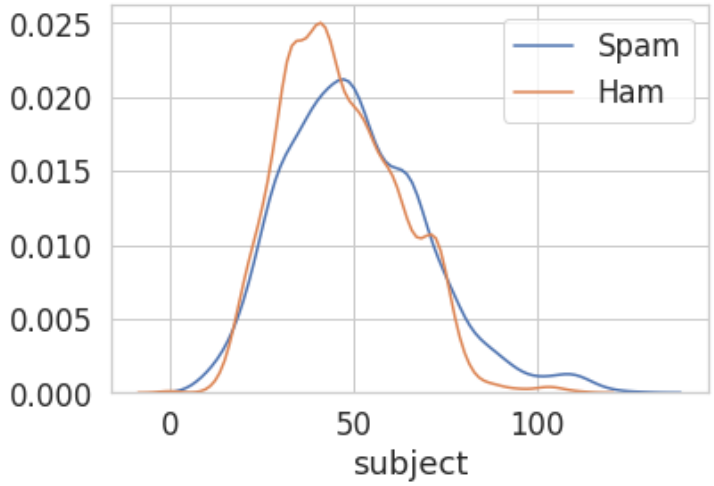
7513 rows × 4 columns

Step 2 Exploratory Data Analysis

I need to first check if the number of cahracters in subjects makes good feature. It turns out that the two distributions of the number of characters in the spam and ham emails have a lot of overlaps. So the number of characters in subjects is not a good feature to use in our model.

In [26]:
```
#number of characters in subjects is not a good feature
sns.distplot(train.query("spam==1")['subject'].str.len(),label='Spam',hist=False)
sns.distplot(train.query("spam==0")['subject'].str.len(),label='Ham',hist=False)
plt.legend()
```

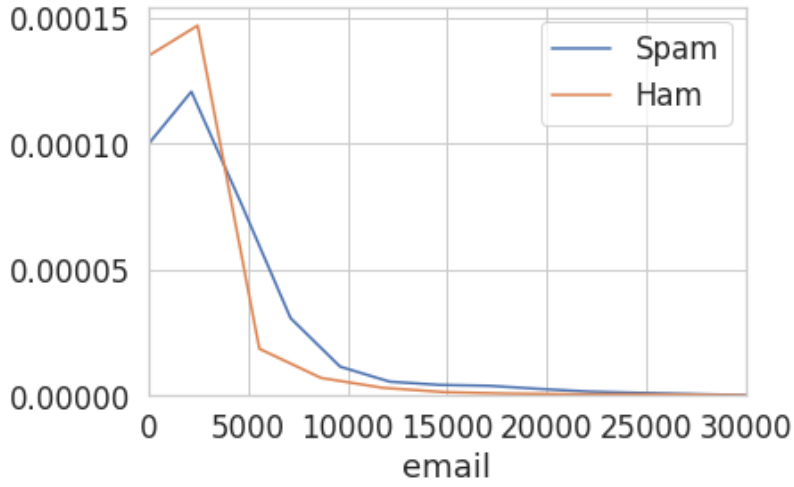
Out[26]: <matplotlib.legend.Legend at 0x7f1cbc0f1eb8>



I then look at the number of characters in the body of the emails. Again the distributions are not clearly separated. This might also not be a good feature.

In [27]:
```
#number of characters in the body is not a good feature
sns.distplot(train.query("spam==1")['email'].str.len(),label='Spam',hist=False)
sns.distplot(train.query("spam==0")['email'].str.len(),label='Ham',hist=False)
plt.xlim(0,30000)
plt.legend()
```

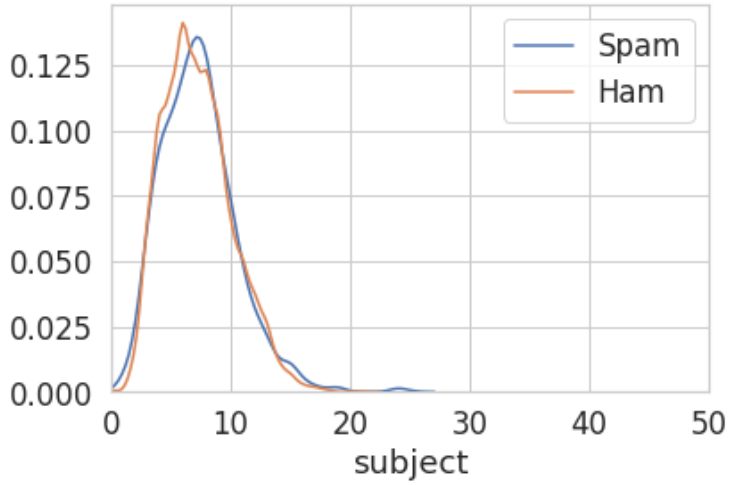
Out[27]: <matplotlib.legend.Legend at 0x7f1cbe80ecf8>



Next, check if the number of words in the subject can be a good feature. Again, the distributions of spam and ham emails are almost identical. I cannot use the number of words in the subject as a feature in our model

In [28]:
```
#number of words in the subject
sns.distplot(train.query("spam==1")['subject'].str.findall('\w+').str.len(),label="Spam",hist=False
sns.distplot(train.query("spam==0")['subject'].str.findall('\w+').str.len(),label="Ham",hist=False)
plt.legend()
plt.xlim(0,50)
```

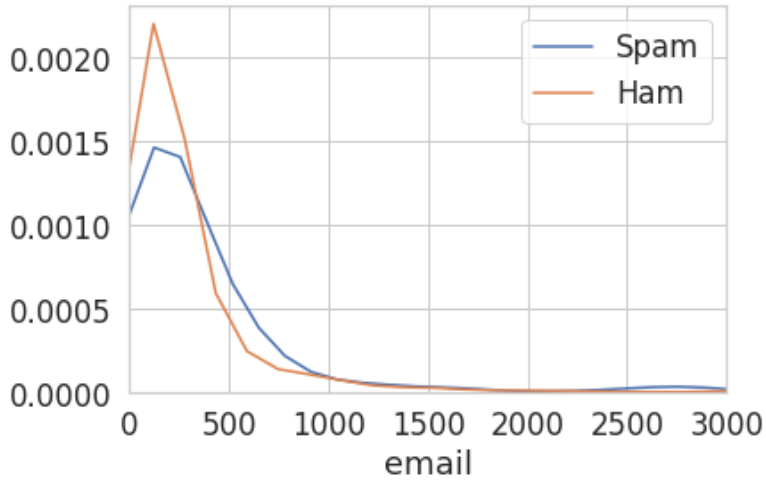
Out[28]: (0, 50)



Same as number of words in the body. Distributions are not clearly separated. Not a good feature to use.

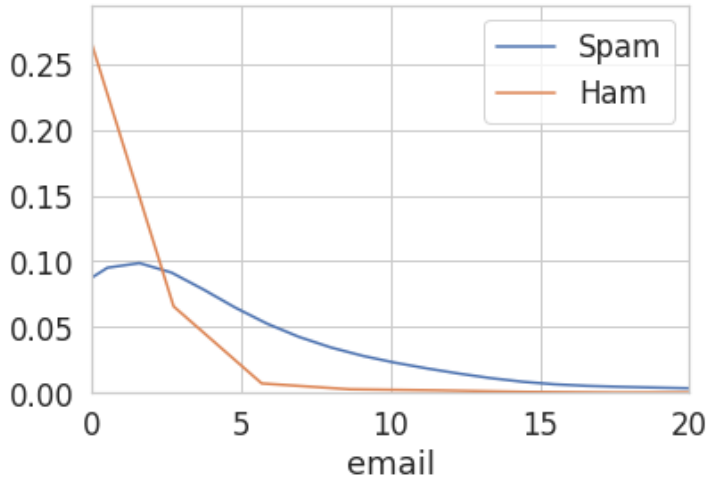
In [29]:
```
#number of words in the body
sns.distplot(train.query("spam==1")['email'].str.findall('\w+').str.len(),label="Spam",hist=False)
sns.distplot(train.query("spam==0")['email'].str.findall('\w+').str.len(),label="Ham",hist=False)
plt.legend()
plt.xlim(0,3000)
```

Out[29]: (0, 3000)

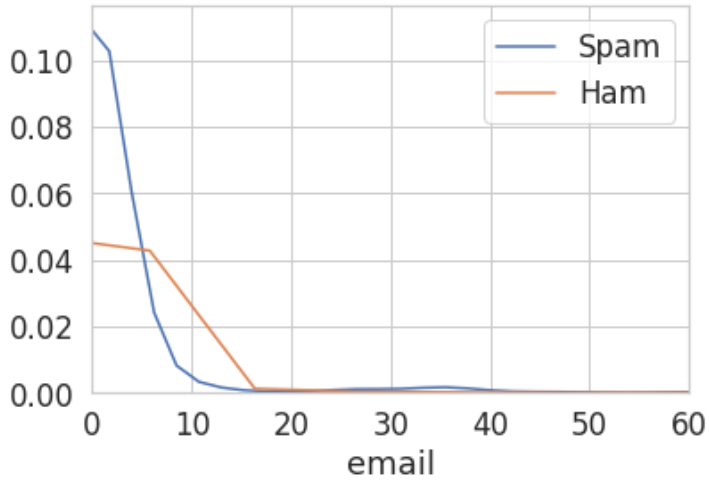


Now I check if the punctuations such as "!" and "#" may be good features to use in our model.

In [30]:
```
#use of ! in email body can be a good one
sns.distplot(train.query("spam==1")['email'].str.findall('!').str.len(),label="Spam",hist=False)
sns.distplot(train.query("spam==0")['email'].str.findall('!').str.len(),label="Ham",hist=False)
plt.legend()
plt.xlim(0,20);
```



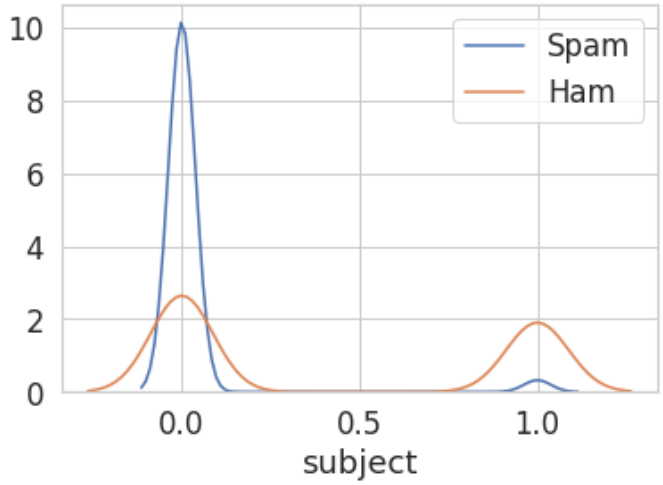
In [31]:
```
sns.distplot(train.query("spam==1")['email'].str.findall('#').str.len(),label="Spam",hist=False)
sns.distplot(train.query("spam==0")['email'].str.findall('#').str.len(),label="Ham",hist=False)
plt.legend()
plt.xlim(0,60);
```



I can use whether an email is a reply or not as a feature. In terms of forward, there is no clear separation between the two distributions, so it does not make a good feature.

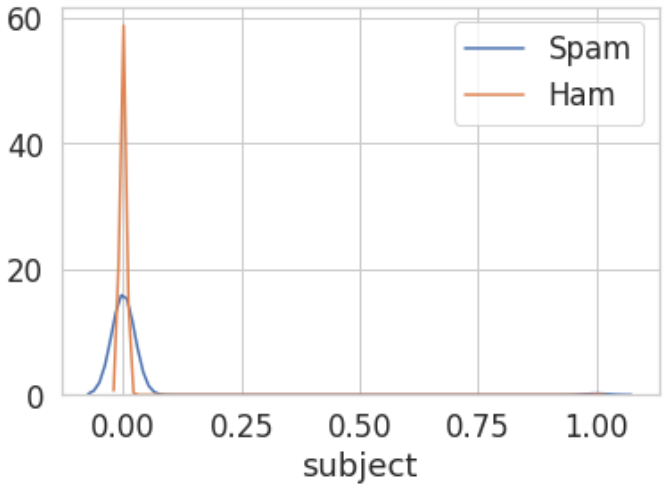
In [32]:
```
#whether an email is reply
sns.distplot(train.query("spam==1")['subject'].str.findall('Subject: Re:').str.len(),label="Spam",h
sns.distplot(train.query("spam==0")['subject'].str.findall('Subject: Re:').str.len(),label="Ham",hi
plt.legend()
```

Out[32]: <matplotlib.legend.Legend at 0x7f1ca58a3da0>



In [33]:
```
#whether an email is forward
sns.distplot(train.query("spam==1")['subject'].str.findall('Subject: Fw').str.len(),label="Spam",hi
sns.distplot(train.query("spam==0")['subject'].str.findall('Subject: Fw').str.len(),label="Ham",his
plt.legend()
```

Out[33]: <matplotlib.legend.Legend at 0x7f1ca56cdbe0>



In the following several blocks of codes, I look for the counts of the 2000 most frequent words in spam and ham emails. No need to include more frequent words than this threshold of 2000 as it will only add some redundant words that only appeared very few times in the data set. I then construct a table to show the counts of each unique words in both the spam and ham emails. I also include the difference between the two counts.

In [34]:
```
#Look for the counts of the 2000 most frequent words in a spam email

s_spam=pd.Series(' '.join(train.query("spam==1")['email']).lower().split()).value_counts()[:2000]
```

In [35]:
```
#Look for the counts of the 2000 frequent words in a ham email
s_ham=pd.Series(' '.join(train.query("spam==0")['email']).lower().split()).value_counts()[:2000]
```

In [36]:
```
#combine the 2000 frequent words from
pd.set_option('display.max_rows',None )
compare=pd.concat([s_spam, s_ham], axis=1, sort=True).rename(columns={0:'count in spam',1:'count in
compare['count diff']=compare['count in spam']-compare['count in ham']
compare.sort_values(by=['count in ham'],ascending=True)
```

Out[36]:

	count in spam	count in ham	count diff
!	178.0	0.0	178.0
individuals	95.0	0.0	95.0
indicate	36.0	0.0	36.0
index	42.0	0.0	42.0
independent	125.0	0.0	125.0
incredible	63.0	0.0	63.0
increases	41.0	0.0	41.0
increased	81.0	0.0	81.0
information!	56.0	0.0	56.0
income.	40.0	0.0	40.0
includes:	44.0	0.0	44.0

In [37]:
```
compare11=compare.loc[compare['count in spam']==0]
compare11.sort_values(by=['count in ham'],ascending=False)
```

http://lists.freshrpms.net/mailman/listinfo/rpm-list	0.0	572.0	-572.0
september	0.0	547.0	-547.0
=09=09=09	0.0	522.0	-522.0
pgp	0.0	510.0	-510.0
%	0.0	500.0	-500.0
global	0.0	496.0	-496.0
story	0.0	479.0	-479.0
groups	0.0	459.0	-459.0
[1]	0.0	456.0	-456.0
red	0.0	435.0	-435.0
rpm	0.0	433.0	-433.0
matthias	0.0	427.0	-427.0
install	0.0	416.0	-416.0

In [38]:
```
compare12=compare.loc[compare['count in ham']==0]
compare12.sort_values(by=['count in spam'],ascending=False)
```

Out[38]:

	count in spam	count in ham	count diff
grants	451.0	0.0	451.0
insurance	450.0	0.0	450.0
fill	398.0	0.0	398.0
orders	343.0	0.0	343.0
fax	317.0	0.0	317.0
dollars	312.0	0.0	312.0
e-mails	302.0	0.0	302.0
income	291.0	0.0	291.0
guide	290.0	0.0	290.0
name:	275.0	0.0	275.0
purchase	252.0	0.0	252.0

In [39]:
```
#compare.sort_values(by=['count in ham'],ascending=False)
```

Lastly and most importantly, I am now ready to construct my model. I need to include features that show big differences in spam and ham emails. I chose some words from the above "compare" dataset, all of which have big count differences in absolute values. Plus, all the words are not prevalent in both email types. I then fit the logistic regression model and used l2 loss penalty and adjusted the updated feature array to according to the cross-validation result to ensure that I didn't overfit the training data.

In [40]:
```
from sklearn.linear_model import LogisticRegression
new_features=['grants','insurance','fill','orders','fax','dollars','e-mails','income','guide','name
              '100%','grant','lose','mortgage','risk','phone','e-mail','mailings','absolutely','emai
              'request','age','business','shipping','valuable','broadcast','earn','yours','$','guara
               'spamassassin','sourceforge','only','transaction','removal','membership','loan','succ
               'transfer','fund','debt','free','ordering','nbsp','color','pour','adult','below','gua
               'url','>>','date','tech',"i'd",'running','cnet','seems','global','story','groups','rp
               'os','signature','seem','pretty','re:','sent:','perhaps','said','issues','invoked','u
               #'2ffont','3b','3e','3cfont','3c','3d','content','transfer',
               #'type','mailing','list','linux','color','card','encoding',
              #'email','address','mailman','one','0d']
x_train=words_in_texts(new_features,train['email'])
y_train=train['spam']
model1=LogisticRegression(penalty='l2')
model1.fit(x_train,y_train)
model1.score(x_train,y_train)
```

Out[40]: 0.9385065885797951

In [55]:
```
#find the accuracy using the model on the validation set
val['email'].str.replace('<[^<]+?>','').str.replace('\n','')

val.reset_index(drop=True,inplace=True)

x_test=words_in_texts(new_features,val['email'])
y_test=val['spam']
model1.score(x_test,y_test)
```

Out[55]: 0.9269461077844311

Create X_train, Y_train datasets for cross validation

In [42]:

```
X_train=pd.DataFrame({'grants':x_train[:,0],'insurance':x_train[:,1],'fill':x_train[:,2],'orders':x_
                      'fax':x_train[:,4],'dollars':x_train[:,5],'e-mails':x_train[:,6],'income':x_t
                      'guide':x_train[:,8],'name':x_train[:,9],'purchase':x_train[:,10],'100%':x_tr
                      'grant':x_train[:,12],'lose':x_train[:,13],'mortgage':x_train[:,14],'risk':x_
                      'phone':x_train[:,16],'e-mail':x_train[:,17],'mailings':x_train[:,18],'absolu
                      'email':x_train[:,20],'money':x_train[:,21],'instead':x_train[:,22],'request'
                      'age':x_train[:,24],'business':x_train[:,25],'shipping':x_train[:,26],'valuab
                      'broadcast':x_train[:,28],'earn':x_train[:,29],'yours':x_train[:,30],'$':x_tr
                      'you':x_train[:,33],'spamassassin':x_train[:,34],'sourceforge':x_train[:,35],
                      'transaction':x_train[:,37],'removal':x_train[:,38],'membership':x_train[:,39
                      'success':x_train[:,41],'improvement':x_train[:,42],'transfer':x_train[:,43],
                      'debt':x_train[:,45],'free':x_train[:,46],'ordering':x_train[:,47],'nbsp':x_t
                      'pour':x_train[:,50],'adult':x_train[:,51],'below':x_train[:,52],'guaranteed!
                      'wrote':x_train[:,54],'url':x_train[:,55],'>>':x_train[:,56],'date':x_train[:
                      'running':x_train[:,60],'cnet':x_train[:,61],'seems':x_train[:,62],'global':x_
                      'story':x_train[:,64],'groups':x_train[:,65],'rpm':x_train[:,66],'java':x_tra
                      'yahoo':x_train[:,68],'os':x_train[:,69],'signature':x_train[:,70],'seem':x_t
                      'pretty':x_train[:,72],'re:':x_train[:,73],'sent:':x_train[:,74],'perhaps':x_
                      'said':x_train[:,76],'issues':x_train[:,77],'invoked':x_train[:,78],'useful':
Y_train=pd.DataFrame({'y train':y_train})
```

Build functions for cross validation

In [43]:

```
def rmse(actual_y, predicted_y):
    """
    Args:
        predicted_y: an array of the prediction from the model
        actual_y: an array of the groudtruth label

    Returns:
        The root mean square error between the prediction and the groudtruth
    """
    return np.sqrt(np.mean((actual_y-predicted_y)**2))
from sklearn.model_selection import KFold

def compute_CV_error(model, X_train, Y_train):
    '''
    Split the training data into 4 subsets.
    For each subset,
        fit a model holding out that subset
        compute the MSE on that subset (the validation set)
    You should be fitting 4 models total.
    Return the average MSE of these 4 folds.

    Args:
        model: an sklearn model with fit and predict functions
        X_train (data_frame): Training data
        Y_train (data_frame): Label

    Return:
        the average validation MSE for the 4 splits.
    '''
    kf = KFold(n_splits=4)
    validation_errors = []

    for train_idx, valid_idx in kf.split(X_train):
        # split the data
        split_X_train, split_X_valid =X_train.iloc[train_idx,:], X_train.iloc[valid_idx,:]
        split_Y_train, split_Y_valid =Y_train.iloc[train_idx], Y_train.iloc[valid_idx]

        # Fit the model on the training split
        model.fit(X=split_X_train,y=split_Y_train)

        # Compute the RMSE on the validation split
        error = rmse(split_Y_valid, model.predict(split_X_valid))


        validation_errors.append(error)

    return np.mean(validation_errors)
```

Now conduct cross-validation to check if I have the optimal number of features

In [44]:
```
errors = []
for N in range(1, X_train.shape[1] + 1):
    #print(f"Trying first {N} features")
    model = model1

    # compute the cross validation error
    error = compute_CV_error(model,X_train.iloc[:,0:N],y_train)

    #print("\tRMSE:", error)
    errors.append(error)

best_num_features = np.argmin(errors)+1
best_err = errors[np.argmin(errors)]

print(f"Best choice, use the first {best_num_features} features")
```

```
Best choice, use the first 80 features
```

After the process of cross-validation, it turns out that we are not overfitting and we should include all the features in our final model.

In [45]:
```
from sklearn.linear_model import LogisticRegression
new_features=['grants','insurance','fill','orders','fax','dollars','e-mails','income','guide','name
              '100%','grant','lose','mortgage','risk','phone','e-mail','mailings','absolutely','emai
              'request','age','business','shipping','valuable','broadcast','earn','yours','$','guara
               'spamassassin','sourceforge','only','transaction','removal','membership','loan','succ
               'transfer','fund','debt','free','ordering','nbsp','color','pour','adult','below','gua
               'url','>>','date','tech',"i'd",'running','cnet','seems','global','story','groups','rp
               'os','signature','seem','pretty','re:','sent:','perhaps','said','issues','invoked','u
               #'2ffont','3b','3e','3cfont','3c','3d','content','transfer',
               #'type','mailing','list','linux','color','card','encoding',
              #'email','address','mailman','one','0d']
x_train=words_in_texts(new_features,train['email'])
y_train=train['spam']
final_model=LogisticRegression(penalty='l2')
final_model.fit(x_train,y_train)
final_model.score(x_train,y_train)
```

Out[45]:
```
0.9385065885797951
```

In [46]:
```
#find the accuracy using the model on the validation set
val['email'].str.replace('<[^<]+?>','').str.replace('\n','')

val.reset_index(drop=True,inplace=True)

x_test=words_in_texts(new_features,val['email'])
y_test=val['spam']
final_model.score(x_test,y_test)
```

Out[46]:
```
0.9281437125748503
```

Question 7: Feature/Model Selection Process

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

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

1.I first used features from my intuition, and the accuracy of the model turned out to be really bad. I then think about what can make a clear division between spam and ham emails. Therefore, I look at words that have big difference in counts and are not

prevalent in both email types.

2. I tried to use punctuations and html tags. But both of them are too generic in both email types.
3. It is surprising to me that some words that I believe must come from spam emails cannot actually be used as a feature because it is too generic and prevalent in both email types.

Question 8: EDA

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

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

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

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

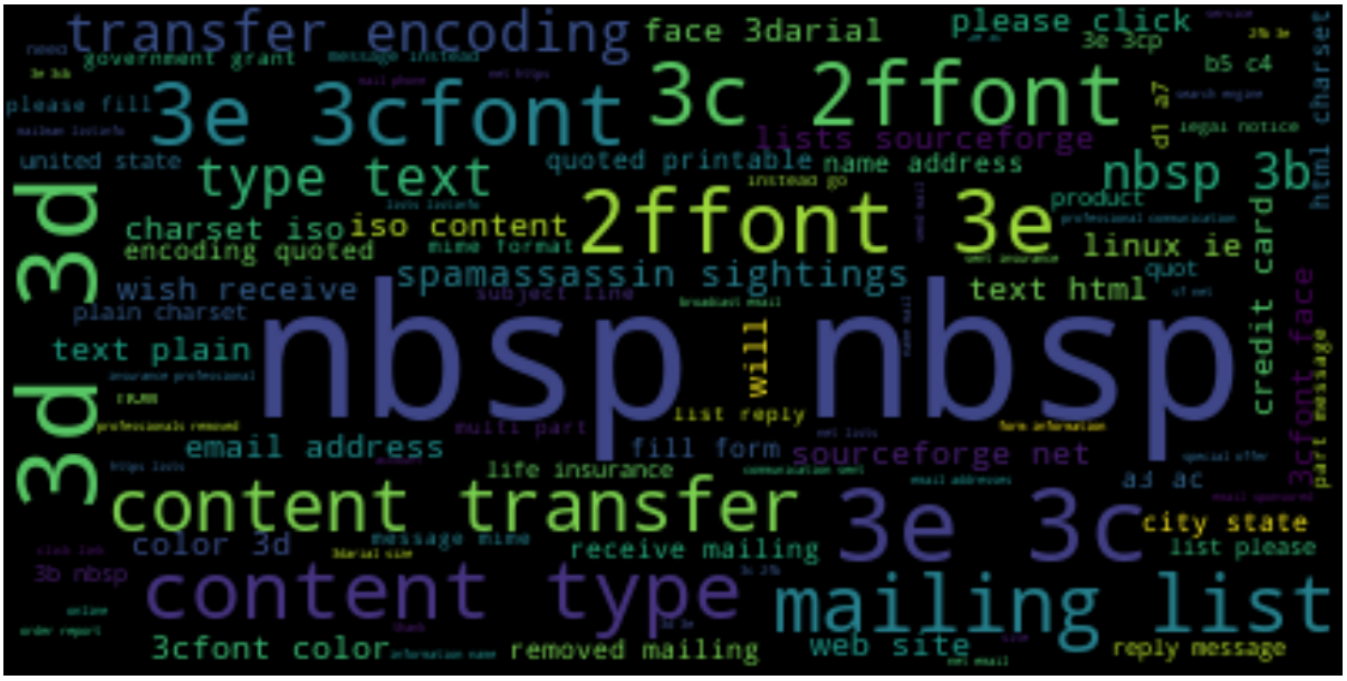
1. Consider the correlation between multiple features (look up correlation plots and `sns.heatmap`).
2. Try to show redundancy in a group of features (e.g. `body` and `html` might co-occur relatively frequently, or you might be able to design a feature that captures all html tags and compare it to these).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

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

In [47]:

```
# Write your description (2-3 sentences) as a comment here:
# The word cloud graph clearly shows the words with the largest counts in the spam emails
# However, we need to be careful because word cloud does not check if a word is prevalent in both em
#

# Write the code to generate your visualization here:

!pip install wordcloud
from os import path
from wordcloud import WordCloud
plt.figure(figsize=(20,20))
spam=train.query('spam==1')['email']
wordcloud=WordCloud().generate(' '.join(spam))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

```
Requirement already satisfied: wordcloud in /srv/conda/envs/data100/lib/python3.6/site-packages
(1.6.0)
Requirement already satisfied: pillow in /srv/conda/envs/data100/lib/python3.6/site-packages (from
wordcloud) (6.2.1)
Requirement already satisfied: numpy>=1.6.1 in /srv/conda/envs/data100/lib/python3.6/site-packages
(from wordcloud) (1.17.4)
Requirement already satisfied: matplotlib in /srv/conda/envs/data100/lib/python3.6/site-packages
(from wordcloud) (3.1.1)
Requirement already satisfied: python-dateutil>=2.1 in /srv/conda/envs/data100/lib/python3.6/site-
packages (from matplotlib->wordcloud) (2.8.1)
Requirement already satisfied: cycler>=0.10 in /srv/conda/envs/data100/lib/python3.6/site-packages
(from matplotlib->wordcloud) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /srv/conda/envs/data100/lib/python3.6/site-pac
kages (from matplotlib->wordcloud) (1.1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /srv/conda/envs/data10
0/lib/python3.6/site-packages (from matplotlib->wordcloud) (2.4.5)
Requirement already satisfied: six>=1.5 in /srv/conda/envs/data100/lib/python3.6/site-packages (fr
om python-dateutil>=2.1->matplotlib->wordcloud) (1.13.0)
Requirement already satisfied: setuptools in /srv/conda/envs/data100/lib/python3.6/site-packages
(from kiwisolver>=1.0.1->matplotlib->wordcloud) (41.6.0)
```



Question 9: ROC Curve

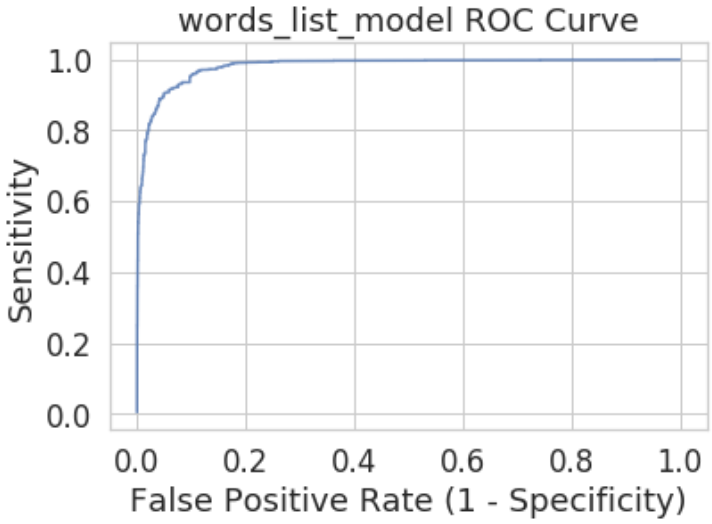
In most cases we won't be able to get no false positives and no false negatives, so we have to compromise. For example, in the case of cancer screenings, false negatives are comparatively worse than false positives — a false negative means that a patient might not discover a disease until it's too late to treat, while a false positive means that a patient will probably have to

take another screening.

Recall that logistic regression calculates the probability that an example belongs to a certain class. Then, to classify an example we say that an email is spam if our classifier gives it ≥ 0.5 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 an ROC curve for your final classifier (the one you use to make predictions for Kaggle). Refer to the Lecture 22 notebook or Section 17.7 of the course text to see how to plot an ROC curve.

In [48]:
```
from sklearn.metrics import roc_curve

# Note that you'll want to use the .predict_proba(...) method for your classifier
# instead of .predict(...) so you get probabilities, not classes
final_model_probabilities = final_model.predict_proba(x_train)[:, 1]
false_positive_rate_values, sensitivity_values, thresholds = roc_curve(y_train, final_model_probabi
plt.step(false_positive_rate_values, sensitivity_values, color='b', alpha=0.8,
         where='post')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('Sensitivity')
plt.title('words_list_model ROC Curve');
```



Question 10: Submitting to Kaggle

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

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

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

You should submit your CSV files to https://www.kaggle.com/c/ds100fa19 (https://www.kaggle.com/c/ds100fa19)

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

In [49]:
```
test['email'].str.replace('<[^<]+?>','').str.replace('\n','')

test.reset_index(drop=True,inplace=True)

x_final_test=words_in_texts(new_features,test['email'])

test_predictions = final_model.predict(x_final_test)
```

In [50]:
```
ok.grade("q10");
```

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    Passed: 3
    Failed: 0
[oooooooooook] 100.0% passed
```

The following saves a file to submit to Kaggle.

In [51]:
```
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 Kaggle for scoring.')
```

```
Created a CSV file: submission_2019-11-25T01:59:10.csv.
You may now upload this CSV file to Kaggle for scoring.
```

Submit

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. **Please save before submitting!**

In [56]:
```
# Save your notebook first, then run this cell to submit.
import jassign.to_pdf
jassign.to_pdf.generate_pdf('proj2.ipynb', 'proj2.pdf')
ok.submit()
```

```
Generating PDF...
Saved proj2.pdf


Saving notebook... Saved 'proj2.ipynb'.
Submit... 100% complete
Submission successful for user: jianghaoliang@berkeley.edu
URL: https://okpy.org/cal/data100/fa19/proj2/submissions/r2o9KK (https://okpy.org/cal/data100/fa19/proj2/submissions/r2o9KK)
```

In [ ]: