Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = "Junsheng Pei"
COLLABORATORS = ""
```

Homework 4: Spam/Ham Classification

Feature Engineering, Logistic Regression, Cross Validation

Due Date: 11/1/18, 11:59PM

Course Policies

Here are some important course policies. These are also located at http://www.ds100.org/fa18/ (http://www.ds100.org/fa18/).

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 solution.

This Assignment

In this homework, 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 minimize 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!

Regarding Submissions - IMPORTANT, PLEASE READ

For this assignment and future assignments (homework and projects) you will also submit your free response and plotting questions to Gradescope. To do this, you can download as PDF; there are two ways to do this:

1. File > Print Preview ----> Save website as PDF
2. Control/Command + P ----> Save website as PDF

If you are having trouble with text being cut off in the generated PDF:

- For cells containing text surrounded in triple quotes (e.g. """ Hello world """), you can press enter in the middle of the string to push the text to a new line so that all the text stays within the box.

You are responsible for submitting and tagging your answers in Gradescope. For each free response and plotting question, please include:

1. Relevant code used to generate the plot or inform your insights
2. The written free response or plot

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

The dataset consists of email messages and their labels (0 for ham, 1 for spam).

Your labeled 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 was spam, 0 if the email was ham (not spam).

The `test` DataFrame contains another set of 1000 unlabeled examples. You will predict labels for these examples 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: Mon Oct 29 20:20:00 2018
MD5 hash of file: 0380c4cf72746622947b9ca5db9b8be8
Using version already downloaded: Mon Oct 29 20:19:59 2018
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

In [4]:
```
test.head()
```

Out[4]:

	id	subject	email
0	0	Subject: CERT Advisory CA-2002-21 Vulnerabilit...	\n \n -----begin pgp signed message-----\n \n ...
1	1	Subject: ADV: Affordable Life Insurance ddbfk\n	low-cost term-life insurance!\n save up to 70%...
2	2	Subject: CAREER OPPORTUNITY. WORK FROM HOME\n	------=_nextpart_000_00a0_03e30a1a.b1804b54\n ...
3	3	Subject: Marriage makes both sexes happy\n	url: http://www.newsisfree.com/click/-3,848315...
4	4	Subject: Re: [SAtalk] SA very slow (hangs?) on...	on thursday 29 august 2002 16:39 cet mike burg...

Question 1a

First let's check if our data contains any nan values. *Fill in the cell below to print whether any of the columns contain nan values.* If there are nan values, replace them with the appropriate filler values. In other words, a nan value in the subject column should be replaced with an empty string.

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

In [5]:
```
# YOUR CODE HERE
#raise NotImplementedError()
if_contain_nan_train = original_training_data.isnull().values.any()
print("Whether original_training_data contains nan values : {0}".format(if_contain_nan_train))
if_contain_nan_test = test.isnull().values.any()
print("Whether test contains nan values : {0}".format(if_contain_nan_test))
# replace nan with '' in original_training_data
original_training_data = original_training_data.fillna('')
```

```
Whether original_training_data contains nan values : True
Whether test contains nan values : True
```

Question 1b

In the cell below, print the text of the first ham and the first spam email in the original training set. Then, discuss one thing you notice that is different between the two that might relate to the identification of spam.

In [6]:
```
# Print the text of the first ham and the first spam emails. Then, fill in your response in the q01 variable:
first_ham = original_training_data[original_training_data['spam'] ==  0].head(1)['email'].values[0]
first_spam = original_training_data[original_training_data['spam'] == 1].head(1)['email'].values[0]
print('first_ham : \n{0}'.format(first_ham))
print('first_spam :\n {0}'.format(first_spam))
q01 = '''
The first ham was written in plain text, however, the first spam was written in html and contains a lot of
scripts in <>
'''
# YOUR CODE HERE
#raise NotImplementedError()
```

```
first_ham :
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/zlfterjnd6jf)
```

```
first_spam :
 <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]:
```
# This is a cell with just a comment but don't delete me if you want to get credit.
```

The first ham was written in plain text, however, the first spam was written in html and contains a lot of scripts in <>

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 validate your model 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. Do not modify this in the following questions, as our assert statements 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 text is ham or spam. This is a *classification* problem, so we can use logistic regression to make a classifier. Recall that to train an logistic regression model we need a numeric feature matrix Φ (pronounced phi as in wifi) and 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 Φ is derived from one email example. Each column of Φ is one feature. 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 world hello']))

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

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.
    '''
    n = len(texts)
    d = len(words)

    indicator_array = np.zeros((n,d))
    for i in range(n):
        for j in range(d):
            if(words[j] in texts.iloc[i]):
                indicator_array[i][j] = 1
    # YOUR CODE HERE
    #raise NotImplementedError()
    return indicator_array
```

In [10]:
```
# If this doesn't error, your function outputs the correct output for this example
assert np.allclose(words_in_texts(['hello', 'bye', 'world'],
                                  pd.Series(['hello', 'hello world hello'])),
                   np.array([[1, 0, 0],
                             [1, 0, 1]]))

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

In [11]:
```
#features_matrix_test_submit = get_feature_matrix(test)
words_in_texts(['hello', 'bye', 'world'], test['email'])
```

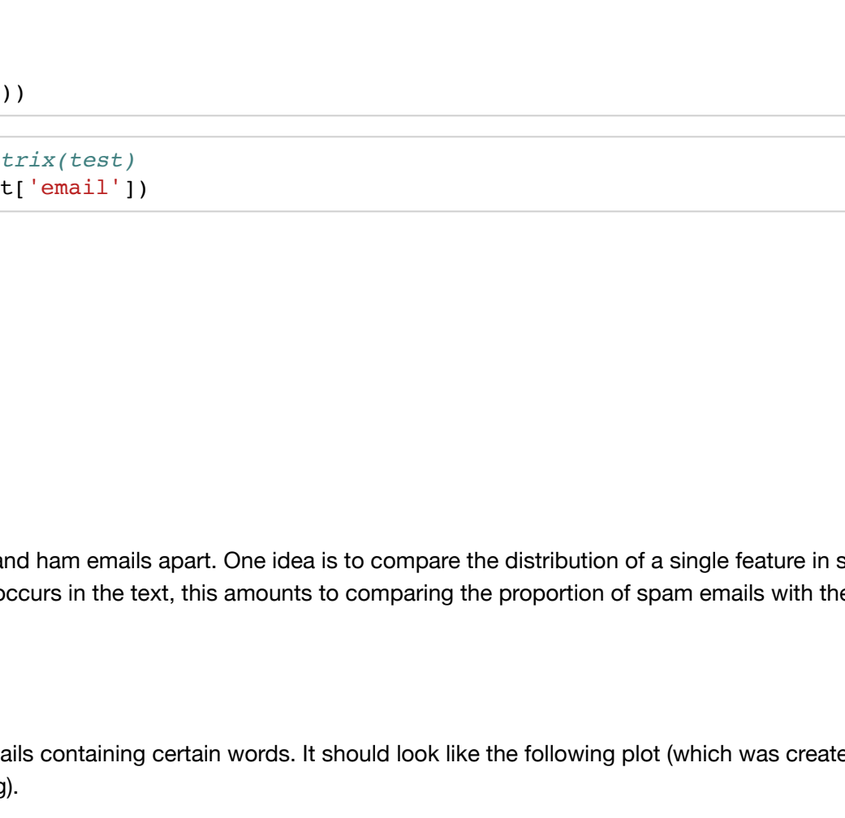
Out[11]:
```
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Basic EDA

Now we need to identify some features that allow us to tell spam and ham emails apart. 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.

Question 3a

Create a bar chart comparing the proportion of spam and ham emails containing certain words. It should look like the following plot (which was created using `sns.barplot`), but you should choose your own words as candidate features. Make sure to use the training set (after splitting).



Hint:

- You can use DataFrame's `.melt` method to "unpivot" a DataFrame. See the following code cell for 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 of each row is a sentence, and the value of 1 or 0 indicates the
display(df)
display(Markdown("> `melt` will turn columns into variale, notice how `word_1` and `word_2` become `variable`, their values are stoed in the value column"))
display(df.melt("type"))
```

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

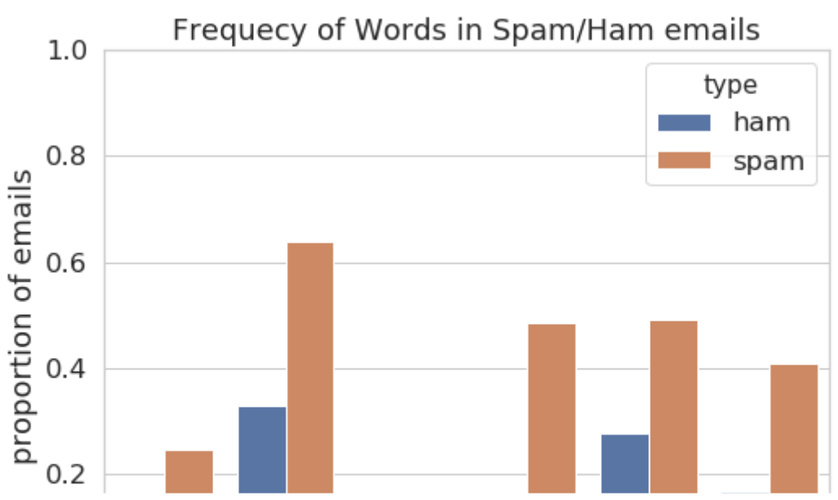
	type	word_1	word_2
0	spam	1	0
1	ham	0	1
2	ham	1	0
3	ham	0	1

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

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

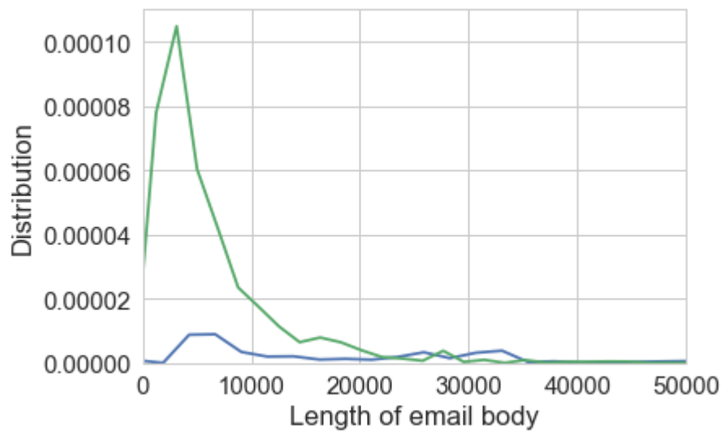
In [13]:
```
train=train.reset_index(drop= True) # We must do this in order to preserve the ordering of emails to labels for words_in_texts
features = ['head', 'br','font','buy','best','free']
#slit the series e-mail by spam and ocunt the word_in_tests
features_matrix = words_in_texts(features ,train['email'])
#convert the np.array to dataframe
features_matrix = pd.DataFrame(features_matrix, columns=features)
# add the spam column
features_matrix['type'] = train['spam'].replace([0,1],['ham','spam'])
#group by 'variable'
grouped_matrix = features_matrix.melt('type').groupby(['variable','type']).mean().reset_index()
#plt
plt.figure(figsize=(8,6))
sns.barplot(x = 'variable', y= 'value', data = grouped_matrix,hue='type')
plt.xlabel('words')
plt.ylim(0,1.0)
plt.ylabel('proportion of emails')
plt.title('Frequecy of Words in Spam/Ham emails')

# YOUR CODE HERE

#raise NotImplementedError()
```

Out[13]:
```
Text(0.5,1,'Frequecy of Words in Spam/Ham emails')
```



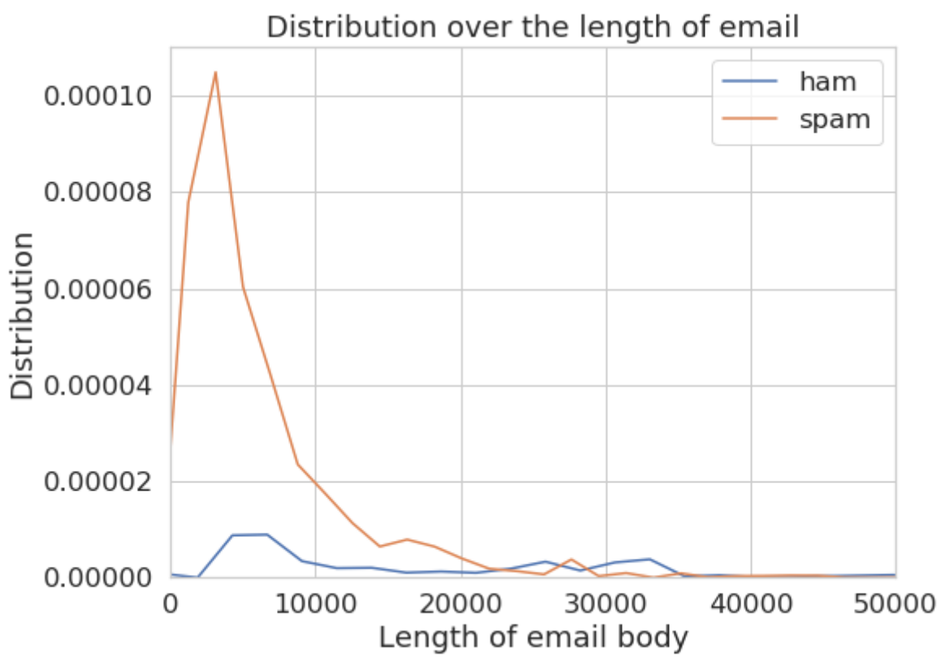
Question 3b

When the feature is binary, it makes sense (as in the previous question) to compare the proportion of 1s in the two classes of email. Otherwise, if the feature can take on many values, it makes sense to compare the distribution under spam to the distribution under ham. Create a *class conditional density plot* like the one below (which was created using `sns.distplot`), comparing the distribution of a feature among all spam emails to the distribution of the same feature among all ham emails. You should use your training set (after splitting). **You may use the length of the email body or create your own feature.** If using length of the email body, please set the xlim to 50000.



In [14]:
```
len_spam = train[train['spam'] == 1]['email'].str.len()
len_ham = train[train['spam'] == 0]['email'].str.len()
plt.figure(figsize=(8,6))
sns.distplot(len_ham,hist=False,label = 'ham')
sns.distplot(len_spam,hist= False,label = 'spam')
plt.xlim(0,50000)
plt.xlabel('Length of email body')
plt.ylabel('Distribution')
plt.title('Distribution over the length of email')
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[14]: Text(0.5,1,'Distribution over the length of email')



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: `Phi_train` and `Y_train`.

`Phi_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.

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

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

# YOUR CODE HERE
#raise NotImplementedError()

Phi_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]:
```
assert np.all(np.unique(Phi_train) == np.array([0, 1]))
assert np.all(np.unique(Y_train) == np.array([0, 1]))
assert Phi_train.shape[0] == Y_train.shape[0]
assert Phi_train.shape[1] == len(some_words)
```

Question 5

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

In [17]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(fit_intercept=True)
lr.fit(X = Phi_train,y = Y_train)
training_accuracy = np.mean(lr.predict(Phi_train) == Y_train)
training_accuracy
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[17]: 0.75762012511646482

In [18]:
```
assert training_accuracy > 0.72
```

Question 6

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 validating on the training set, which may lead to a misleading accuracy measure, especially if we used the training set to identify discriminative features. In future parts of this analysis, it will be safer to hold out some of our data for model validation and comparison.

Presumably, our classifier will be used for **filtering**, i.e. preventing messages labeled `spam` from reaching someone's inbox. Since we are trying 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.

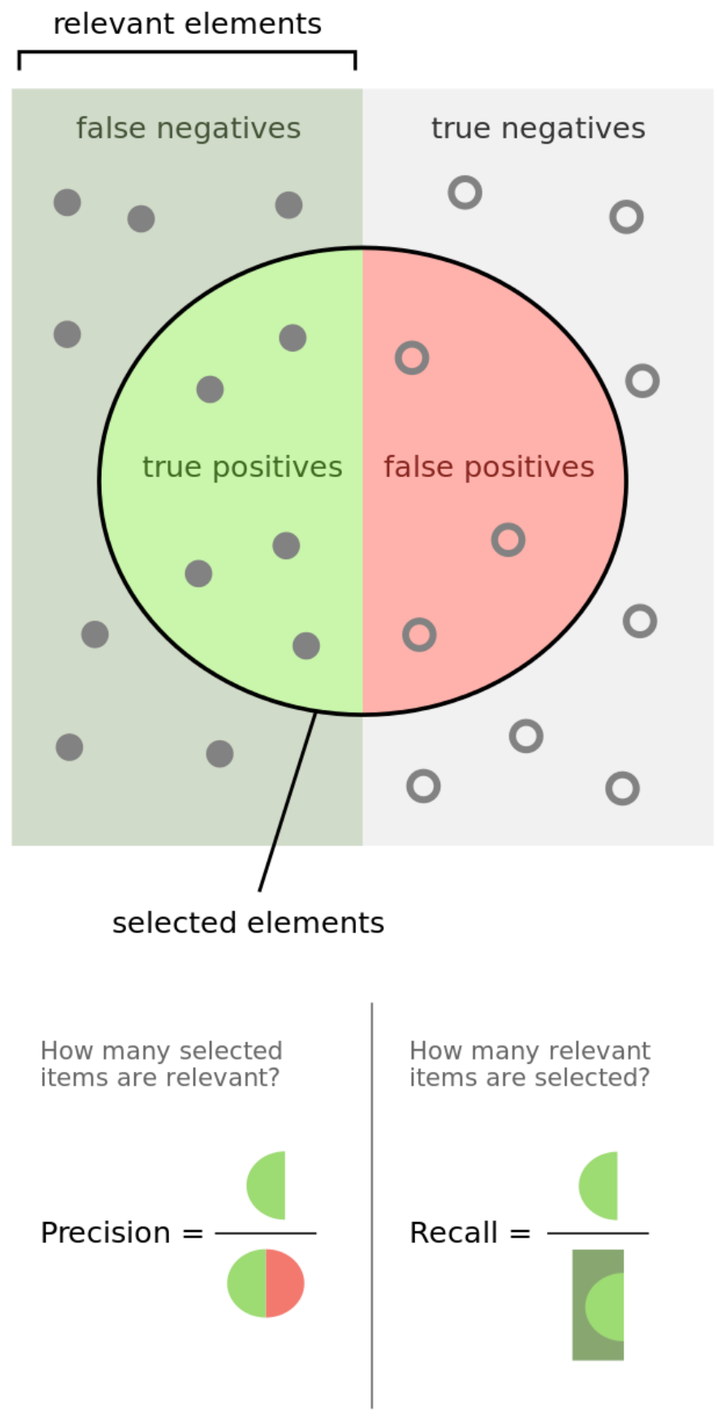
These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

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. Answer the following questions in the cells below:

- (a) Suppose we have a classifier that just predicts 0 (ham) for every email. How many false positives are there? How many false negatives are there? Provide specific numbers using the training data from Question 4.
- (b) Suppose we have a classifier that just predicts 0 (ham) for every email. What is its accuracy on the training set? What is its recall on the training set?
- (c) What are the precision, recall, and false-alarm rate of the logistic regression classifier in Question 5? Are there more false positives or false negatives?
- (d) Our logistic regression classifier got 75.6% prediction accuracy (number of correct predictions / total). How does this compare with predicting 0 for every email?
- (e) Given the word features we gave you above, name one reason this classifier is performing poorly.
- (f) Which of these two classifiers would you prefer for a spam filter and why? (N.B. there is no "right answer" here but be thoughtful in your reasoning).

In [19]:
```
# provide number of FP and FN, respectively,
# for a classifier that always predicts 0 (never predicts positive...)
zero_predictor_fp = 0
zero_predictor_fn = len(train[train['spam'] == 1])
print('false positives : {0}'.format(zero_predictor_fp))
print('false negatives : {0}'.format(zero_predictor_fn))

# YOUR CODE HERE
#raise NotImplementedError()
```

```
false positives : 0
false negatives : 1918
```

In [20]:
```
# This is a cell with just a comment but don't delete me if you want to get credit.
```

In [21]:
```
# provide training accuracy & recall, respectively,
# for a classifier that always predicts 0
zero_predictor_acc = 0
zero_predictor_recall = 0
print('zero_predictor_acc : {0}'.format(zero_predictor_acc))
print('zero_predictor_recall : {0}'.format(zero_predictor_recall))

# YOUR CODE HERE
#raise NotImplementedError()
```

```
zero_predictor_acc : 0
zero_predictor_recall : 0
```

In [22]:
```
# This is a cell with just a comment but don't delete me if you want to get credit.
```

In [23]:
```
# provide training accuracy & recall, respectively,
# for logistic regression classifier from question 5
predicted = lr.predict(Phi_train)
TP = len(Y_train[Y_train == predicted][Y_train == 1])
TN = len(Y_train[Y_train == predicted][Y_train == 0])
FP = len(Y_train[Y_train != predicted][Y_train == 0])
FN = len(Y_train[Y_train != predicted][Y_train == 1])
logistic_predictor_precision = TP/(TP + FP)
logistic_predictor_recall = TP/(TP + FN)
logistic_predictor_far = FP/(FP+TN)
print('FN : {0} , FP:{1}'.format(FN,FP))
print('precision : {0}, recall : {1}, far :{2}'.format(logistic_predictor_precision,logistic_predictor_recall,logistic_predictor_far))
# or we can use confusion_matrix, the results are the same
# YOUR CODE HERE
#raise NotImplementedError()
```

```
FN : 1699 , FP:122
precision : 0.6422287390029325, recall : 0.11418143899895725, far :0.021805183199285077
```

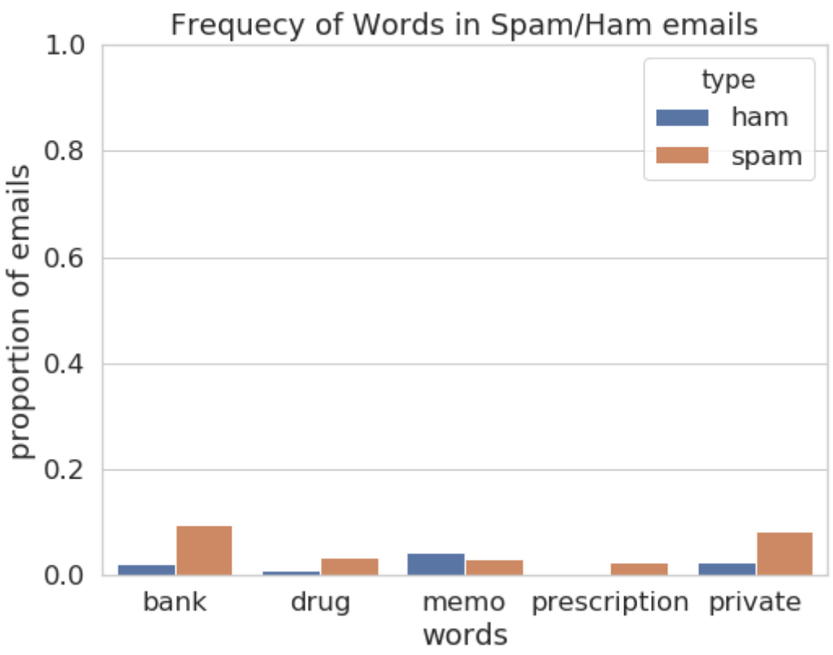
In [24]:
```
# This is a cell with just a comment but don't delete me if you want to get credit.
```

In [25]:
```
#part (d)
print(' prediction accuracy: {0}'.format(len(train[train['spam'] == 0]) /len(train)))
```

```
prediction accuracy: 0.7447091707706642
```

In [26]:
```
#part (e) to answer this question, we see the frequency of each word
#convert the np.array to dataframe
features_matrix = pd.DataFrame(Phi_train, columns=some_words)
# add the spam column
features_matrix['type'] = train['spam'].replace([0,1],['ham','spam'])
#group by 'variable'
grouped_matrix = features_matrix.melt('type').groupby(['variable','type']).mean().reset_index()
#plt
plt.figure(figsize=(8,6))
sns.barplot(x = 'variable', y= 'value', data = grouped_matrix,hue='type')
plt.xlabel('words')
plt.ylim(0,1.0)
plt.ylabel('proportion of emails')
plt.title('Frequecy of Words in Spam/Ham emails')
```

Out[26]: Text(0.5,1,'Frequecy of Words in Spam/Ham emails')



(a) Since every email will be classified as negative, there are 0 false positive. And the number of false negative will be the number of spams in training set with is 1918.

(b) Since every email will be classified as negative, TP will be 0. precision = $\frac{TP}{TP+FP} = 0$, recall = $\frac{TP}{TP+FN} = 0$

(c) precision : 0.6422, recall : 0.1142, far :0.0218. And there are more false negatives

(d) The prediction accuracy of predicting 0 is 74.5%, only 1% worse than the logistic regression. So the logistic regression didn't make any improvement.

(e) From the plot above, the reason might be that the frequency of each word is too low. Another reason might be that the number of the features are not enough.

(f)If we only have feature words which don't help us distinct spam and ham, we will preper zero classfication, because logistic regression has a higher false-alarm rate, and zero classfication is faster. But if we have some feature words are useful like body, offer, html, I prefer logistic regression.

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.

To prevent you from overfitting to the test set, you may only upload predictions to Kaggle twice per day. This means you should start early and rely on your **validation data** to estimate your Kaggle scores.

Here are some ideas for improving your model:

1. Finding better features based on the email text. Some example features are:
   A. Number of characters in the subject / body
   B. Number of words in the subject / body
   C. Use of punctuation (e.g., how many '!' were there?)
   D. Number / percentage of capital letters
   E. Whether the email is a reply to an earlier email or a forwarded email
2. Finding better words to use as features. Which words are the best at distinguishing emails? This requires digging into the email text itself.
3. Better data processing. For example, many emails contain HTML as well as text. You can consider extracting out the text from the HTML to help you find better words. Or, you can match HTML tags themselves, or even some combination of the two.
4. Model selection. You can adjust parameters of your model (e.g. the regularization parameter) to achieve higher accuracy. Recall that you should use cross-validation to do feature and model selection properly! Otherwise, you will likely overfit to your training data.

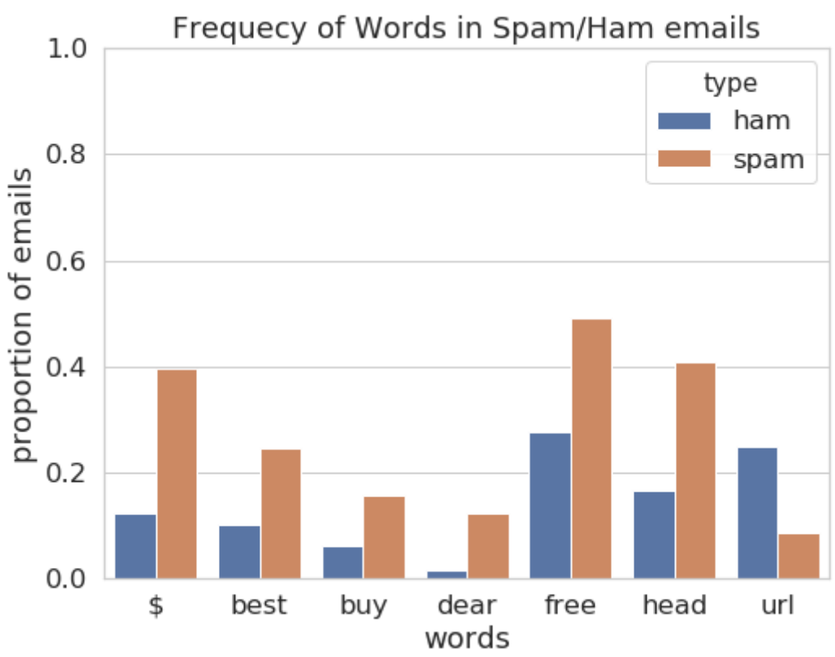
You may use whatever method you prefer in order to create features. However, **you are only allowed to train logistic regression models and their regularized forms**. This means no random forest, k-nearest-neighbors, neural nets, etc.

We will not give you a code skeleton 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.*

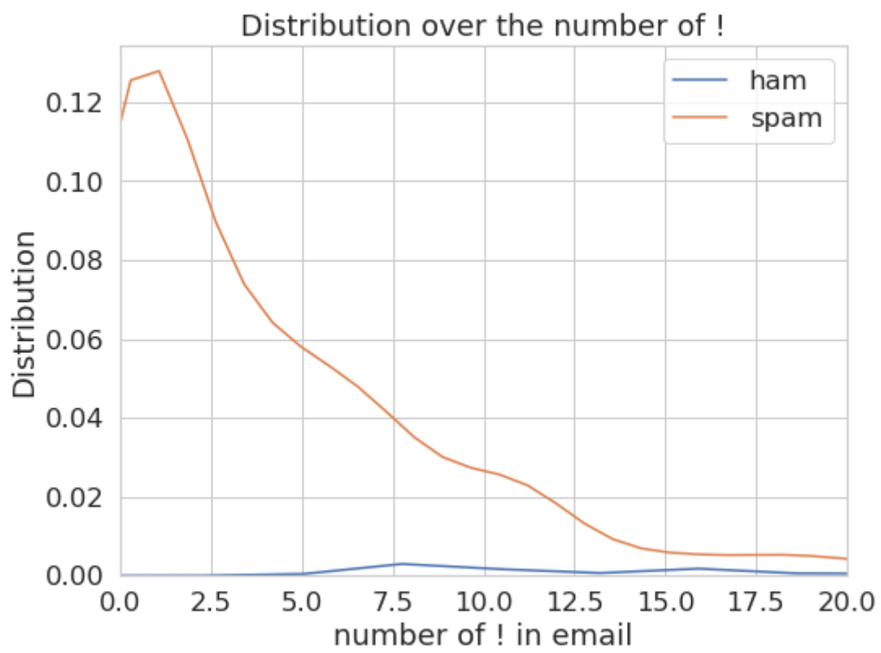
In [27]:
```
# choose some typical feature words that can disticnt spam and ham

train=train.reset_index(drop= True) # We must do this in order to preserve the ordering of emails to labels for words_in_texts
features = ['head', 'dear','url','buy','best','free','$',]
#slit the series e-mail by spam and ocunt the word_in_tests
features_matrix = words_in_texts(features ,train['email'])
#convert the np.array to dataframe
features_matrix = pd.DataFrame(features_matrix, columns=features)
# add the spam column
features_matrix['type'] = train['spam'].replace([0,1],['ham','spam'])
#group by 'variable'
grouped_matrix = features_matrix.melt('type').groupby(['variable','type']).mean().reset_index()
#plt
plt.figure(figsize=(8,6))
sns.barplot(x = 'variable', y= 'value', data = grouped_matrix,hue='type')
plt.xlabel('words')
plt.ylim(0,1.0)
plt.ylabel('proportion of emails')
plt.title('Frequecy of Words in Spam/Ham emails')

# YOUR CODE HERE

#raise NotImplementedError()
```

Out[27]: Text(0.5,1,'Frequecy of Words in Spam/Ham emails')



In [28]:
```
# find some features that behave very differntly in ham and spam

# len_spam = train[train['spam'] == 1]['email'].str.split().apply(len)
# len_ham = train[train['spam'] == 0]['email'].str.split().apply(len)
# len_spam = train[train['spam'] == 1]['email'].str.count('>')
# len_ham = train[train['spam'] == 0]['email'].str.count('>')
len_spam = train[train['spam'] == 1]['email'].str.count('!')
len_ham = train[train['spam'] == 0]['email'].str.count('!')

plt.figure(figsize=(8,6))
sns.distplot(len_ham,hist=False,label = 'ham')
sns.distplot(len_spam,hist= False,label = 'spam')
plt.xlim(0,20)
plt.xlabel('number of ! in email')
plt.ylabel('Distribution')
plt.title('Distribution over the number of !')
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[28]: Text(0.5,1,'Distribution over the number of !')



In [29]:
```
#choose the meaningful feature words by the number of occurence in spam and ham

spam_word = pd.Series(' '.join(train[train['spam'] == 1]['email']).split()).value_counts()[:100]
ham_word = pd.Series(' '.join(train[train['spam'] == 0]['email']).split()).value_counts()[:100]
in_spam = np.setdiff1d(np.array(spam_word.index),np.array(ham_word.index))
in_ham = np.setdiff1d(np.array(ham_word.index),np.array(spam_word.index))
print(in_spam)
print(in_ham)
```

```
['&' '<br>' '<div' '<input' '<meta' '<option' '<p' '<p><font' '<tr>=20'
 'address' 'arial,' 'border=3d"0"' 'business' 'click' 'e-mail' 'email'
 'face="verdana"><font' 'face="verdana,' 'face=3d"arial,' 'free'
 'helvetica,' 'how' 'information' 'make' 'money' 'order' 'over' 'people'
 'please' 'receive' 'report' 'sans-serif"' 'send' 'time' 'us' 'want'
 'width=3d"100%"']
[' <a' '*' '--' '-->'
 '//////////////////////////////////////////////////////////////////////////'
 '2002' '<!--' '<tr' '<tr><td' 'alt=""' 'been' 'border="0"' 'but'
 'cellspacing="0"' "don't" 'face="arial,' 'he' 'helvetica"' "it's" 'its'
 'like' 'other' 'some' 'src="http://home.cnet.com/b.gif"'
 'src="http://www.cnet.com/b.gif"' 'src="http://www.zdnet.com/b.gif"'
 'than' 'their' 'there' 'they' 'use' 'when' 'which' 'width="1"' 'width=1'
 'would' '|']
```

In [30]:
```
# get the feature matrix, the input X should be a dataframe

def get_feature_matrix(X):
    # choose the feature words by digging into the email
    features_words = ['head','body','offer','html','please','url','$',
        'free','business','receive','after',
       '&', '<br>', '<div', '<input', '<meta', '<option', '<p', '<p><font',
       '<tr>=20', 'address', 'arial,', 'border=3d"0"', 'click',
       'e-mail', 'email', 'face="verdana"><font', 'face="verdana,',
       'face=3d"arial,', 'helvetica,', 'how', 'information',
       'make', 'money', 'order', 'over', 'people',
       'report', 'sans-serif"', 'send', 'time', 'us', 'want',
       'width=3d"100%"',' <a', '*', '--', '-->',
       '2002', '<!--', '<tr', '<tr><td', 'alt=""', 'been', 'border="0"',
       'but', 'cellspacing="0"', "don't", 'face="arial,', 'he',
       'helvetica"', "it's", 'its', 'like', 'other', 'some',
        'than', 'their', 'there',
       'they', 'use', 'when', 'which', 'width="1"', 'width=1', 'would', '|']
    # add more useful features
    features_matrix= words_in_texts(features_words ,X['email'])

    numberOfwords = np.array(X['email'].str.split().apply(len)).reshape(len(X),1)
    lengthOfemail = np.array(X['email'].str.len() ).reshape(len(X),1)
    contain_Re  = np.array(X['subject'].str.contains('Re')).astype(float).reshape(len(X),1)
    contain_Fw  = np.array(X['subject'].str.contains('Fw')).astype(float).reshape(len(X),1)
    number_right  = np.array(X['email'].str.count('>')).reshape(len(X),1)
    number_of_punctions  = np.array(X['email'].str.count('!')).reshape(len(X),1)
    contain_now = np.array(X['subject'].str.contains('now')).astype(float).reshape(len(X),1)
    contain_free  = np.array(X['subject'].str.contains('Free')).astype(float).reshape(len(X),1)
    #number_of_Cap  = np.array(X['email'].str.findall(r'[A-Z]').str.len()).reshape(len(X),1)
    #number_left  = np.array(X['email'].str.count('<')).reshape(len(X),1)

    features_matrix = np.hstack((features_matrix,lengthOfemail,
                                 number_right,number_of_punctions,contain_now,
                                contain_Fw+contain_Re,numberOfwords,contain_free))
    features_matrix = np.nan_to_num(features_matrix)
    return features_matrix
```

In [31]:
```
#apply crossvalidation to choose a proper regularition parameter
# and we find _lambda = 0.5 is the best which means C =2
from sklearn.model_selection import KFold

def compute_CV_error(model, X_train, Y_train):
    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[train_idx], X_train[valid_idx]
        split_Y_train, split_Y_valid = Y_train[train_idx], Y_train[valid_idx]

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

        # Compute the RMSE on the validation split
        error = np.mean(Y_predic == split_Y_valid)

        validation_errors.append(error)

    return np.mean(validation_errors)
regulation_parameter = [0.2,0.33,0.5,1,2,3,5,8]

errors = []
features_matrix_cv = get_feature_matrix(train)
for _lambda in regulation_parameter:
    print("Trying regulation_parameter:", _lambda)
    model= LogisticRegression(fit_intercept = True, C = _lambda)
    error = compute_CV_error(model, features_matrix_cv, train['spam']) # compute the cross validation error
    print("\n error:", error)
    errors.append(error)

```

```
Trying regulation_parameter: 0.2

 error: 0.94768958632
Trying regulation_parameter: 0.33

 error: 0.947689798859
Trying regulation_parameter: 0.5

 error: 0.948754690739
Trying regulation_parameter: 1

 error: 0.949153910068
Trying regulation_parameter: 2

 error: 0.950484900937
Trying regulation_parameter: 3

 error: 0.947423133099
Trying regulation_parameter: 5

 error: 0.947290296144
Trying regulation_parameter: 8

 error: 0.947689798859
```

In [32]:
```
lr_final = LogisticRegression(fit_intercept = True, C = 2)
features_matrix_train = get_feature_matrix(train)
lr_final.fit(X = features_matrix_train ,y = Y_train)
training_accuracy = np.mean(lr_final.predict(features_matrix_train) == Y_train)

features_matrix_test = get_feature_matrix(val)
test_accuracy = np.mean(lr_final.predict(features_matrix_test) == val['spam'])
print('train accuracy: {0}, test accuracy: {1}'.format(training_accuracy, test_accuracy))
# add the spam column
```

```
train accuracy: 0.955144416345002, test accuracy: 0.9461077844311377
```

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. Followed by the instruction in question 6, I looked into the following feature:
    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
F. Others like the number of > in email, whether 'now' or 'Free' occur in subject I plot the frequncy of each feature as we did in question 4. And see whether those feature are very different in spam and ham, if they are quite different, I add these feature into the feature matrix to see whether they help improve the feature.

Then I try to find more meaningful feature words by find the most 100 frequency word in spam and ham respectively, and get the difference set for the two sets, which contain words in spam100 but not in ham100 or words in ham100 but not in spam100. Then we check theres words, drop some strange words and add them to the feature matrix

2. I find adding more meaningful feature words are extremly useful, like html, free, offer,'S', which have a high frequency in spam but not in ham. And features like Number of characters in the body, Use of punctuation and Whether the email is a reply are also useful we can see the distinct frequency in spam and ham by ploting the distribution line. But the features like Number / percentage of capital letters may not be very useful because the distrubution lines are quite similar.

3. I find normally the more features you add to the feature matrix, the better accuracy you will get, but if the number of features are too large, the improvement is not obvious, and sometimes it will make the accuracy decrease. And some features related to html language are good features.

Question 8 (EDA)

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

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

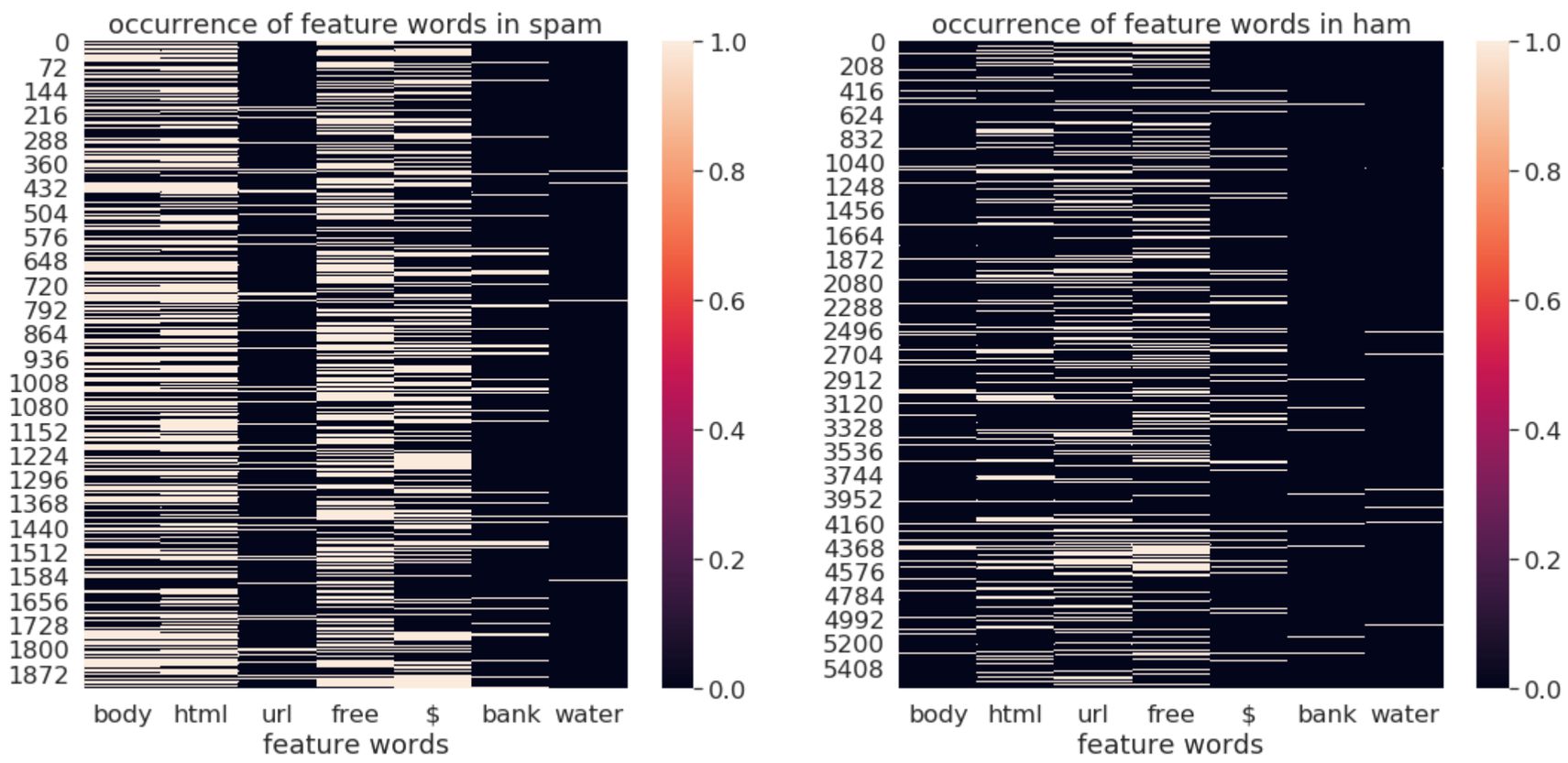
Feel to create as many plots as you want in your process of feature selection, but select one for the cells below.

You should not show us a visualization just like in 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. Use a word-cloud or another visualization tool to characterize the most common spam words.
4. Visually depict whether spam emails tend to be wordier (in some sense) than ham emails.

In [33]:
```
# YOUR CODE HERE
features = ['body', 'html','url','free','$','bank','water']
#slit the series e-mail by spam and ocunt the word_in_tests
plt.figure(figsize=(18,8))
plt.subplot(1,2,1)
features_matrix_spam = words_in_texts(features ,train[train['spam'] == 1]['email'])
sns.heatmap(features_matrix_spam)
plt.xticks(np.arange(len(features))+0.5,features)
plt.title('occurrence of feature words in spam')
plt.xlabel('feature words')
plt.subplot(1,2,2)
features_matrix_ham = words_in_texts(features ,train[train['spam'] == 0]['email'])
sns.heatmap(features_matrix_ham)
plt.xticks(np.arange(len(features))+0.5,features)
plt.title('occurrence of feature words in ham')
plt.xlabel('feature words')
#raise NotImplementedError()
```

Out[33]:
```
Text(0.5,43.5,'feature words')
```



From the heatmap we plot above, we can see the occurence of each feature word in ham and spam. We find 'body','html','free','$',occur much more frequently in spam than in ham. And 'url'occur much more frequently in ham than in spam. So they are all good feature words. But 'bank' and 'word' both occur infrequently in spam and ham, they are not good feature words.

Besides we can see the correlation between some words, like 'body' and html, that often occur at the same time.

Question 9 (Making a Precision-Recall Curve)

We can trade off between precision and recall. In most cases we won't be able to get both perfect precision (i.e. no false positives) and recall (i.e. 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.

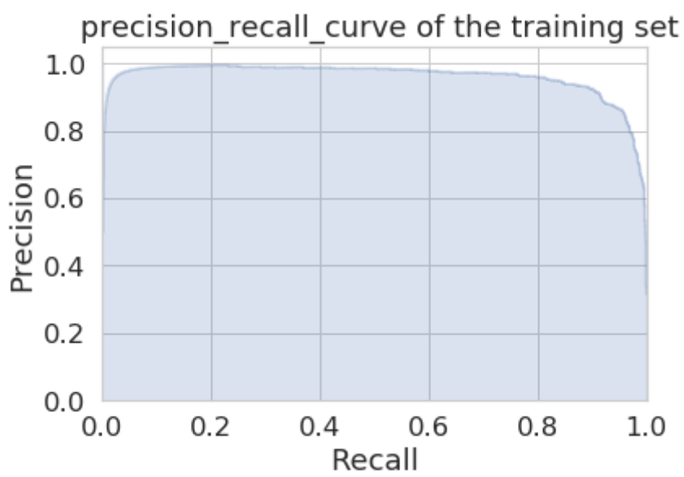
The precision-recall curve shows this trade off for each possible cutoff probability. In the cell below, plot a precision-recall curve (http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html#plot-the-precision-recall-curve) for your final classifier (the one you use to make predictions for Kaggle).

In [34]:
```
features_matrix_train.shape
```

Out[34]:
```
(7513, 84)
```

In [35]:
```
from sklearn.metrics import precision_recall_curve
from sklearn.utils.fixes import signature

# Note that you'll want to use the .predict_proba(...) method for your classifier
# instead of .predict(...) so you get probabilities, not classes
y_score = lr_final.predict_proba(features_matrix_train)
precision, recall, thresholds = precision_recall_curve(train['spam'], y_score[:,1])
step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})
plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title(' precision_recall_curve of the training set')
# YOUR CODE HERE
#raise NotImplementedError()
```

Out[35]: Text(0.5,1,' precision_recall_curve of the training set')



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 grade for this part 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/t/d9a7013e7fd048c291ff7efe6e1ac25e (https://www.kaggle.com/t/d9a7013e7fd048c291ff7efe6e1ac25e)

In [36]:
```
# CHANGE ME (Currently making random predictions)
features_matrix_test_submit = get_feature_matrix(test)
test_predictions = lr_final.predict(features_matrix_test_submit)

# YOUR CODE HERE
#raise NotImplementedError()
```

In [37]:
```
# must be ndarray of predictions
assert isinstance(test_predictions, np.ndarray)

# must be binary labels (0 or 1) and not probabilities
assert np.all((test_predictions == 0) | (test_predictions == 1))

# must be the right number of predictions
assert test_predictions.shape == (1000, )
```

In [38]:
```
# Please do not modify this cell
```

The following saves a file to submit to Kaggle.

In [39]:
```
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.

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

# must be binary labels (0 or 1) and not probabilities
assert np.all((test_predictions == 0) | (test_predictions == 1))

# must be the right number of predictions
assert test_predictions.shape == (1000, )

# 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_2018-11-08T01:21:25.csv.
You may now upload this CSV file to Kaggle for scoring.
```

Submission

You're done!

Before submitting this assignment, ensure to:

1. Restart the Kernel (in the menubar, select Kernel->Restart & Run All)
2. Validate the notebook by clicking the "Validate" button

Finally, make sure to **submit** the assignment via the Assignments tab in Datahub