In [1]:

```
# Initialize OK
from client.api.notebook import Notebook
ok = Notebook('proj2b.ok')
```

```
=====================================================================
Assignment: proj2b
OK, version v1.14.20
=====================================================================
```

Project 2 Part B: Spam/Ham Classification

Classifiers

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 Project 2 Part A, you made an effort to understand the data through EDA, and did some basic feature engineering. You also built a Logistic Regression model to classify Spam/Ham emails. In Part B, you will learn how to evaluate the classifiers you built. You will also have the chance to improve your model by selecting more features.

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
6a	1
6b	1
6c	2

Question	Points
6d	2
6e	1
6f	3
7	6
8	6
9	15
Total	37

Setup

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)
```

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()

from sklearn.model_selection import train_test_split

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

```
Using version already downloaded: Fri Apr 24 14:47:12 2020
MD5 hash of file: 0380c4cf72746622947b9ca5db9b8be8
Using version already downloaded: Fri Apr 24 14:47:11 2020
MD5 hash of file: a2e7abd8c7d9abf6e6fafc1d1f9ee6bf
```

The following code is adapted from Part A of this project. You will be using it again in Part B.

In [4]:

```
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 = 1 * np.array([texts.str.contains(word) for word in words]).T
    return indicator_array

some_words = ['drug', 'bank', 'prescription', 'memo', 'private']

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

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

Out[4]:

```
(array([[0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0],
        [0, 0, 0, 1, 0]]),
 array([0, 0, 0, 0, 0], dtype=int64))
```

Recall that you trained the following model in Part A.

In [5]:

```
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
```

Evaluating Classifiers

The model you trained 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.

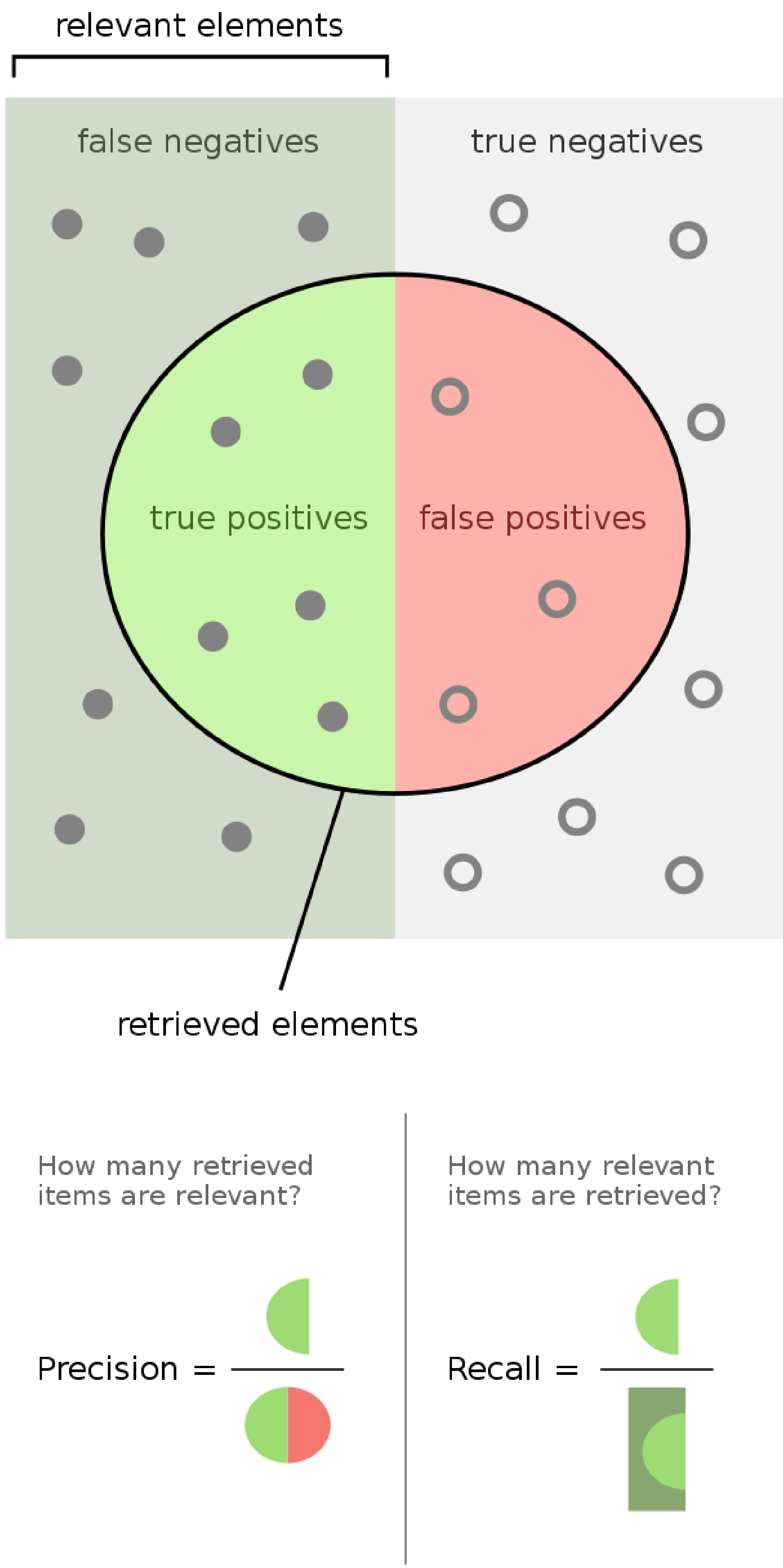
These definitions depend both on the true labels and the predicted labels. False positives and false negatives may be of differing importance, leading us to consider more ways of evaluating a classifier, in addition to overall accuracy:

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

Recall measures the proportion $\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$ of spam emails that were correctly flagged as spam.

False-alarm rate measures the proportion $\frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$ 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 [6]:

```
zero_predictor_fp = 0
zero_predictor_fn = Y_train.sum()

print('Zero false positive:', zero_predictor_fp)
print('Zero false negative:', zero_predictor_fn)
```

```
Zero false positive: 0
Zero false negative: 1918
```

In [7]:

```
ok.grade("q6a");
```

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

---------------------------------------------------------------------
Test summary
    Passed: 2
    Failed: 0
[ooooooooook] 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 [8]:

```
zero_predictor_acc = np.mean(Y_train==0)
zero_predictor_recall = 0

print('Zero accuracy:', zero_predictor_acc)
print('Zero recall:', zero_predictor_recall)
```

```
Zero accuracy: 0.7447091707706642
Zero recall: 0
```

In [9]:

```
ok.grade("q6b");
```

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

---------------------------------------------------------------------
Test summary
    Passed: 2
```

```
    Failed: 0
[ooooooooook] 100.0% passed
```

Question 6c

Provide brief explanations of the results from 6a and 6b. Explain why the number of false positives, number of false negatives, accuracy, and recall all turned out the way they did.

Write your answer here, replacing this text.

- Since the zero_predictor never gives 1:
  - There's no "positive" and therefore the number of false positives is 0
  - The number of false negatives would be the number of positive observations in the data
- Based on the justification above:
  - The correct predictions would be all 0's in the dataset, so precision would be the number of 0's divided by the total number of observations
  - Since the zero_predictor doesn't recall anything, the recall ratio is 0

Question 6d

Compute the precision, recall, and false-alarm rate of the `LogisticRegression` classifier created and trained in Part A. Do **NOT** use any `sklearn` functions.

Note: In lecture we used the `sklearn` package to compute the rates. Here you should work through them using just the definitions to help build a deeper understanding.

In [10]:

```
Y_train_hat = model.predict(X_train)

TP = sum((Y_train_hat == Y_train) & (Y_train_hat == 1))
TN = sum((Y_train_hat == Y_train) & (Y_train_hat == 0))
FP = sum((Y_train_hat != Y_train) & (Y_train_hat == 1))
FN = sum((Y_train_hat != Y_train) & (Y_train_hat == 0))

logistic_predictor_precision = TP/(TP+FP)
logistic_predictor_recall = TP/(TP+FN)
logistic_predictor_far = FP/(FP+TN)

print('Logistic false positives:', FP)
print('Logistic false negatives:', FN)
```

```
Logistic false positives: 122
Logistic false negatives: 1699
```

In [11]:

```
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 Part A?

Write your answer here, replacing this text.

As shown above, there are more false negatives (1699) than false positives (122)

Question 6f

1. Our logistic regression classifier got 75.8% 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.

Write your answer here, replacing this text.

1. The accuracy of the zero_predictor is around 74.5%, so the logistic regression classifier performs better, but not too much
2. One potential reason would be that there are too many zeros in the features matrix (X_{train}). In other words, many emails don't have the words we choose, and therefore these words can't explain the variation between the emails without these words.
3. I would prefer to use the zero_predictor:
   - As for the logistic regression classifier, there are too many false negatives. In other words, many ham emails are filtered.
   - This is a bad practice because looking for ham emails in the spam box seems to be more painful than simply remove the spam emails that are not successfully filtered

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 9).

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.*

Question 7: Feature/Model Selection Process

In the 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?

Write your answer here, replacing this text.

1. To find better features:

    - Count features:
        - I extract the length of the email body, the number of occurance of "rich format" (such as html tags and hashtags).
        - In this cases, the number matters, so I include them as counts rather than 0-1's
        - The count numbers are very different across variables. This may contaminate the optimization algorithm, so I also normalize the count features
    - 0-1 dummy features:

- I extract whether the email is replied or forwarded, and one-hot encoding of bag of words for subjects and email bodies
- The choice process of bag of words is tricky. I basically read the distinguished features of spam emails, and compare the chance of occurance in ham/spam of each word, repeatedly

2. What did you try that worked / didn't work?

- I have tried to find the number of capital letters, an accurate description of "format richness", etc. All these procedures work, and I include them in my final model
- Instead of whether a word occurs or not (words_in_texts), I tried to encode them as counts. However, this makes the model's prediction accuracy worse

3. What was surprising in your search for good features?

- Including more features (in particular, words) doesn't necessarily improve the model performance. This is probably because many emails don't have my chosen words, and those chosen words can't explain whether a email is ham/spam if such email doesn't have those words

In [12]:

```
# reload the dataset to resume the capital letters in body

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

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

In [13]:

```
train.loc[train['spam']==0,['subject','email']].sample(10)
```

Out[13]:

	subject	email
1245	Subject: Re: [zzzzteana] An announcement\n	\n > Mr Tim Chapman, freelance gentleman of le...
6018	Subject: Re: [Razor-users] Stripping the SpamA...	At 1:52 PM -0400 8/13/02, Theo Van Dinter wrot...
6655	Subject: Dawn raids stoke fires of resentment\n	URL: http://www.newsisfree.com/click/-2,865571...
1166	Subject: Mplayer\n	Hey\n \n Since I upgraded to redhat8 mplayer -...
1347	Subject: Re: Goodbye Global Warming\n	\n ----- Original Message ----- \n From: "John...
6875	Subject: Hopes fade in Ulster crisis talks\n	URL: http://www.newsisfree.com/click/-2,865570...
4119	Subject: [use Perl] Stories for 2002-09-20\n	use Perl Daily Newsletter\n \n In this issue:\...
5350	Subject: Education debate\n	URL: http://www.newsisfree.com/click/215,11,21...
7314	Subject: Re: Quick php advice needed :-)\n	I am a php programmer (very busy one at that -...
7596	Subject: Apple Switch parodies\n	URL: http://www.askbjoernhansen.com/archives/2...

In [14]:

```
train.loc[train['spam']==1,['subject','email']].sample(10)
```

Out[14]:

	subject	email

	subject	email
35	Subject: Set & Forget! Blast Your Ad Over 200 ...	(See Disclaimer below)\n \n \n Dear Internet M...
312	Subject: The database that Bill Gates doesnt w...	IMPORTANT NOTICE: Regarding your domain name\...
2343	Subject: Internet ve Para \n	CashIC.com =DDnternetten para kazandiran, en o...
1937	Subject: Incredible Pictures!!!!!!!!\n	<html>\n <body>\n <p>Do you like Sexy Animals ...
5988	Subject: Hgh: safe and effective release of yo...	<html>\n <head>\n <meta http-equiv=3D"Conte...
1604	Subject: [ILUG] STOP THE MLM INSANITY\n	Greetings!\n \n You are receiving this letter ...
936	Subject: attached data that can be stored for ...	FIND PROSPECTS FOR YOUR BIZ/PRODUCTS...FAST !!...
6767	Subject: ADV: Promote Your Website!\n	Removal instructions below\n \n \n I saw your ...
6121	Subject: don't proscrastinate...it's only $14....	------=_NextPart_000_00B6_07E34C7A.C3030C43\n ...
6468	Subject: BUSINESS ASSISTANCE\n	\n Dear Sir.\n \n First, I must solicit your c...

In [15]:

```
train.loc[train['spam']==0,'email'].str.count(r'(<\/?(html|body|title|head|font|meta|ta
```

Out[15]:

```
count    5595.000000
mean       45.017337
std       207.860672
min         0.000000
25%         0.000000
50%         0.000000
75%         2.000000
max      4867.000000
Name: email, dtype: float64
```

In [16]:

```
train.loc[train['spam']==1,'email'].str.count(r'(<\/?(html|body|title|head|font|meta|ta
```

Out[16]:

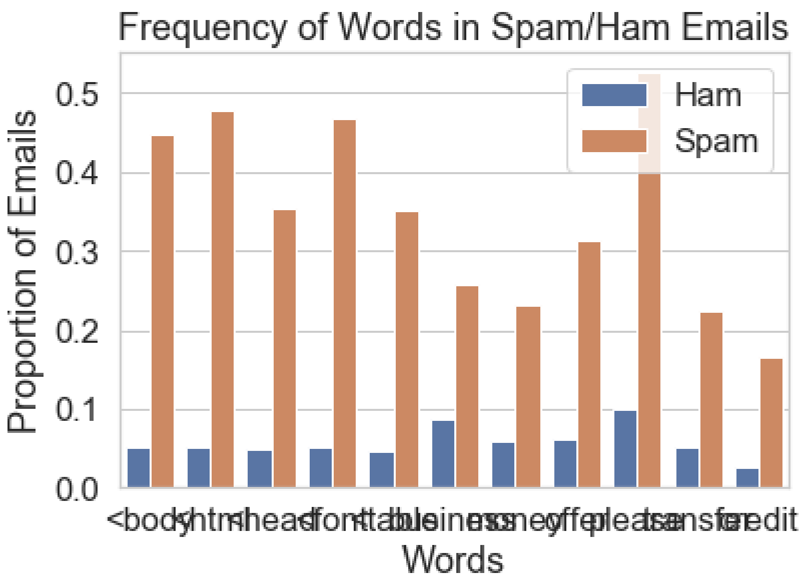
```
count    1918.000000
mean      116.115746
std       284.009573
min         0.000000
25%         1.000000
50%        28.000000
75%       148.000000
max      8111.000000
Name: email, dtype: float64
```

In [17]:

```
# enter the bags of words
body_words = ['<body','<html','<head','<font','<table','business','money','offer','plea
subject_words = ['business','money','\$','member','card','free','credit','account','loa
```

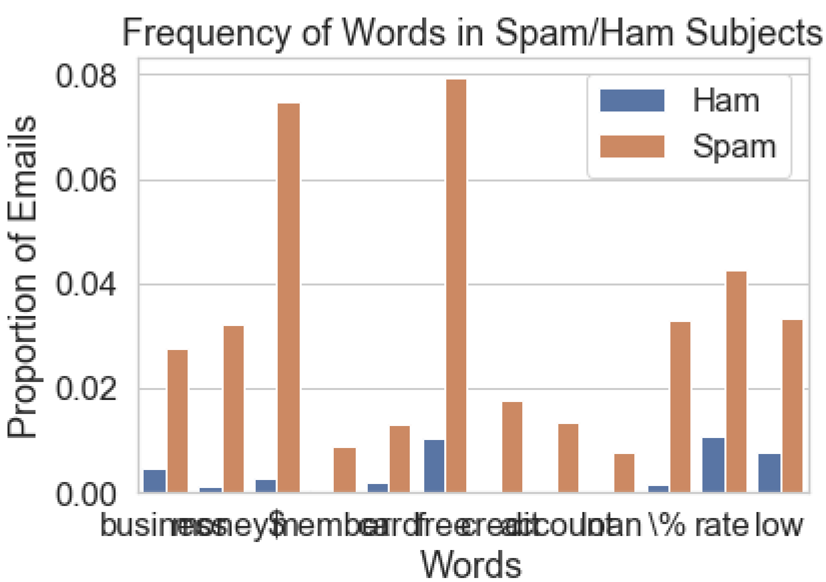
In [18]:

```
train=train.reset_index(drop=True) # We must do this in order to preserve the ordering
temp = pd.DataFrame(words_in_texts(body_words, train['email'].str.lower()),columns=body
temp['spam'] = train['spam'].replace({0:'Ham', 1:'Spam'})
sns.barplot(data=temp.melt('spam'), x='variable', y='value', hue='spam', ci=None)
plt.xlabel('Words')
plt.ylabel('Proportion of Emails')
plt.legend()
plt.title('Frequency of Words in Spam/Ham Emails');
```



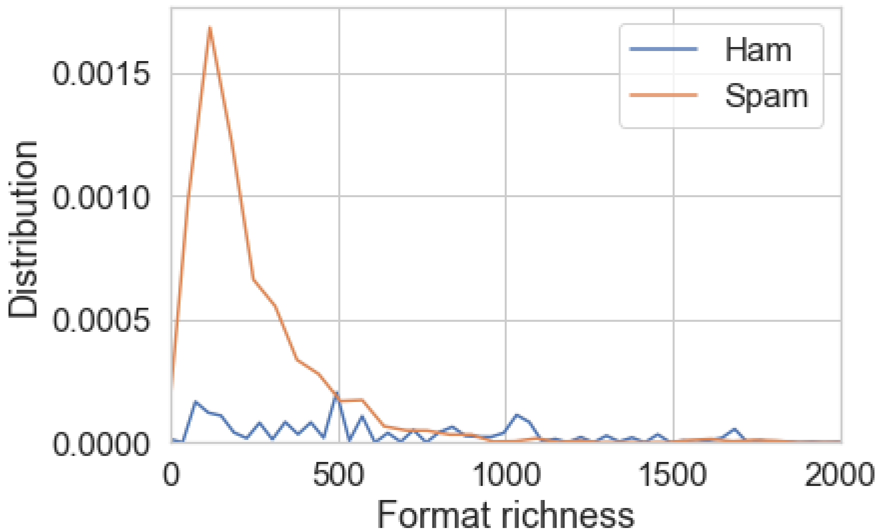
In [19]:

```
train=train.reset_index(drop=True) # We must do this in order to preserve the ordering
temp = pd.DataFrame(words_in_texts(subject_words, train['subject'].str.lower()),columns
temp['spam'] = train['spam'].replace({0:'Ham', 1:'Spam'})
sns.barplot(data=temp.melt('spam'), x='variable', y='value', hue='spam', ci=None)
plt.xlabel('Words')
plt.ylabel('Proportion of Emails')
plt.legend()
plt.title('Frequency of Words in Spam/Ham Subjects');
```



In [20]:

```
sns.distplot(train.loc[train['spam']==0,'email'].str.count(r'(<\/?(html|body|title|head
sns.distplot(train.loc[train['spam']==1,'email'].str.count(r'(<\/?(html|body|title|head
plt.legend()
plt.xlim(0,2000)
plt.xlabel('Format richness')
plt.ylabel('Distribution');
```



In [21]:

```
def features(data, body_words, subject_words):
    '''
    usage: feed in data and bags of words, return features
    '''
    # initialize
    X_data = pd.DataFrame()

    # count data
    X_data['body_length'] = data['email'].str.len()
    X_data['rich_format'] = data['email'].str.count(r'(<\/?(html|body|title|head|font|m
    X_data['body_cap'] = data['email'].str.count(r'[A-Z]').fillna(0)
    X_data['subject_num'] = data['subject'].str.count(r'\d').fillna(0)
    X_data['punctuation'] = data['subject'].str.count('!').fillna(0)
    X_data['subject_cap'] = (data['subject'].str.count(r'[A-Z]').fillna(0))

    # normalize count data
    X_data = (X_data - X_data.mean())/X_data.std()

    # one-hot encoding
    X_data['re_fwd'] = data['subject'].str.lower().str.contains('fwd:|re:|fw:').fillna(
    X_data = pd.concat([
                    X_data.reset_index(drop=True),
                    pd.DataFrame(words_in_texts(body_words, data['email'].str.lower()))
                    pd.DataFrame(words_in_texts(subject_words, data['subject'].str.lowe
                       ],1).fillna(0).to_numpy()
    return X_data
```

In [22]:

```
# create features and labels for training and validation data

X_train = features(train, body_words, subject_words)
X_val = features(val, body_words, subject_words)

Y_val = np.array(val['spam'])
```

In [23]:

```
# fit the training data

from sklearn.linear_model import LogisticRegressionCV
```

```
model = LogisticRegressionCV(max_iter=500)
model.fit(X_train, Y_train)

print('Training accuracy:', round(model.score(X_train, Y_train),3))
print('Validation accuracy:', round(model.score(X_val, Y_val),3))
```

```
Training accuracy: 0.923
Validation accuracy: 0.915
```

In [24]:

```
# fit the entire training set

X_final = features(original_training_data, body_words, subject_words)
X_test = features(test, body_words, subject_words)

Y_final = np.array(original_training_data['spam'])
```

In [25]:

```
# fit the entire dataset using the best model so far
model.fit(X_final, Y_final)

print('Final model accuracy:', round(model.score(X_final, Y_final),3))
```

```
Final model accuracy: 0.922
```

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:

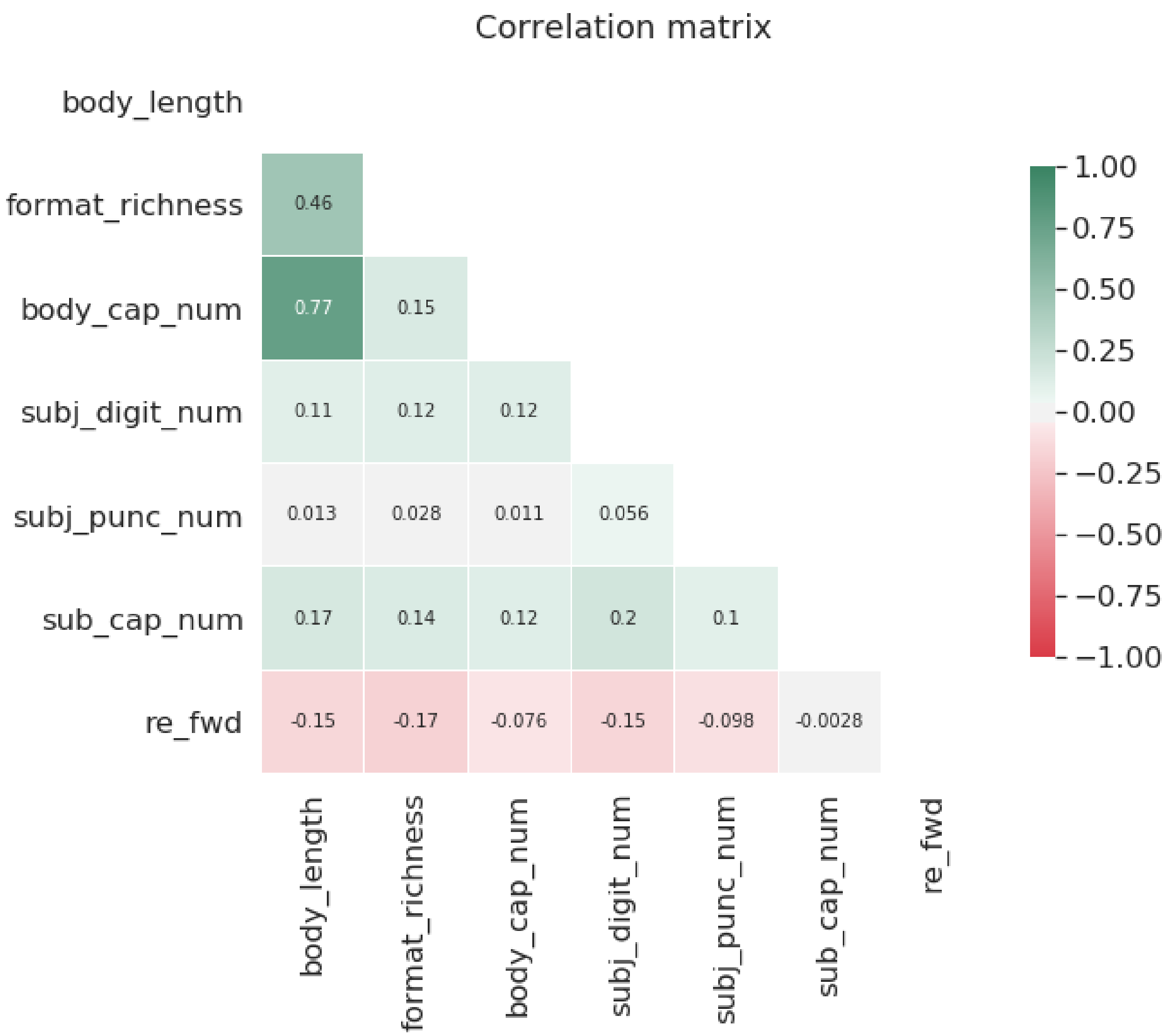
1. Consider the correlation between multiple features (look up correlation plots and `sns.heatmap`).
2. Try to show redundancy in a group of features (e.g. `body` and `html` might co-occur relatively frequently, or you might be able to design a feature that captures all html tags and compare it to these).
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 [26]:

```
# Write your description (2-3 sentences) as a comment here:
#
```

```
# I include a correlation matrix of my count features.
# The number of capital letters in the body is highly positively correlated
# with the body length.
# This makes sense: an email with longer body should have more sentences and
# therefore more capital letters.
# The "format richness" is also highly positively correlated with the body length.
# The same logic may apply: an email with longer body should tend to include more
# hashtags, equal signs, etc.

# Write the code to generate your visualization here:
EDA = pd.DataFrame(X_train[:,:7], columns=['body_length', 'format_richness', 'body_cap_
                                     'subj_digit_num', 'subj_punc_num', 'sub_cap_num',

colormap = sns.diverging_palette(10, 150, n=100, as_cmap=True)

corr = EDA.corr()
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

fig, ax = plt.subplots(figsize=(9, 10))

sns.heatmap(corr, mask=mask, cmap=colormap, vmin=-1, vmax=1,
            center=0, square=True, linewidths=.1, cbar_kws={"shrink": .5}, annot = True

plt.title('Correlation matrix');

# Note: if your plot doesn't appear in the PDF, you should try uncommenting the followi
# plt.show()
```



Question 9: 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/t/c76d80f7d3204159865a324ec2936f18

Note: You may submit up to 4 times a day. If you have submitted 4 times on a day, you will need to wait until the next day for more submissions.

Note that this question is graded on an absolute scale based on the accuracy your model achieves on the test set and the score does not depend on your ranking on Kaggle.

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

In [27]:

```
test_predictions = model.predict(X_test)
```

In [28]:

```
ok.grade("q9");
```

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

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

The following saves a file to submit to Kaggle.

In [29]:

```
from datetime import datetime

# Assuming that your predictions on the test set are stored in a 1-dimensional array ca
# test_predictions. Feel free to modify this cell as long you create a CSV in the right

# 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_2020-04-23T01:13:14.csv.
You may now upload this CSV file to Kaggle for scoring.
```