Notebook

April 16, 2019

0.0.1 Question 1c

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

There seems to be a lot of html mark-up formatting in spam emails, indicating more of a webpage display rather than plain text. Also the subject matter of the spam appears to be attempting to sell a product.

0.0.2 Question 3a

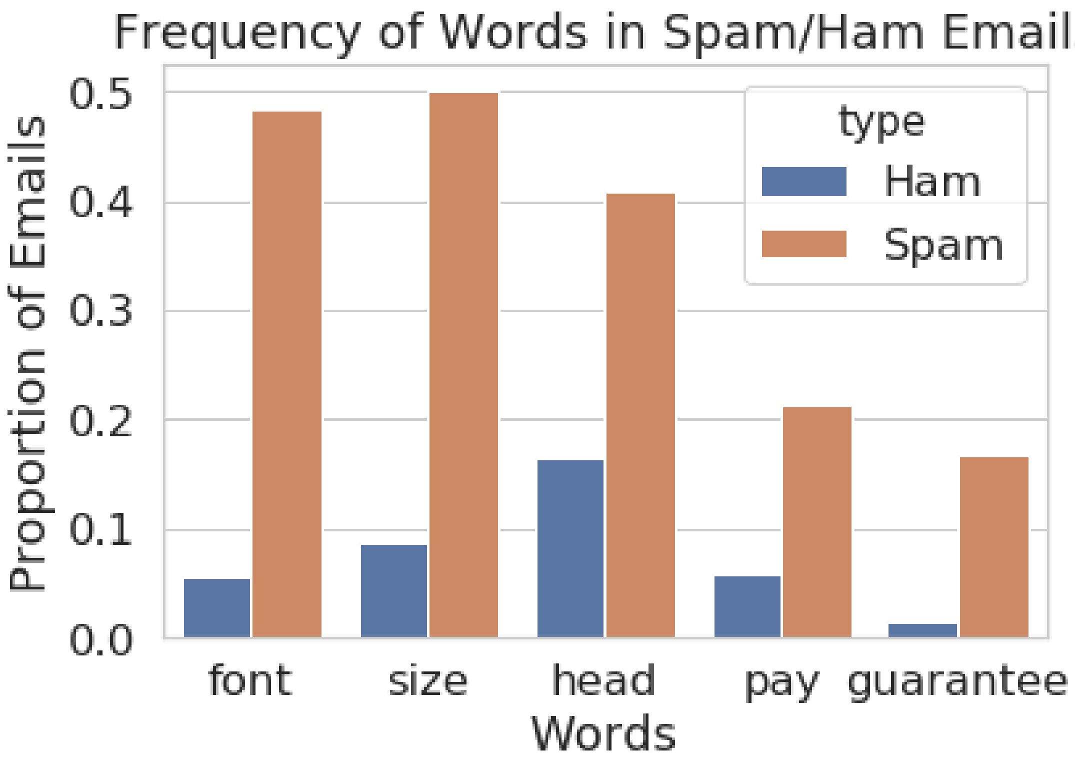
Create a bar chart like the one above comparing the proportion of spam and ham emails containing certain words. Choose a set of words that are different from the ones above, but also have different proportions for the two classes. Make sure to only consider emails from `train`.

```
In [12]: train=train.reset_index(drop=True) # We must do this in order to preserve the ordering of emai

         occ = words_in_texts(['font', 'size', 'head', 'pay', 'guarantee'], train['email'])

         df = pd.DataFrame({
             'font': occ.T[0],
             'size': occ.T[1],
             'head': occ.T[2],
             'pay': occ.T[3],
             'guarantee': occ.T[4],
             'type': train['spam'].replace({0: 'Ham', 1: 'Spam'}).values
         }).melt("type")

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



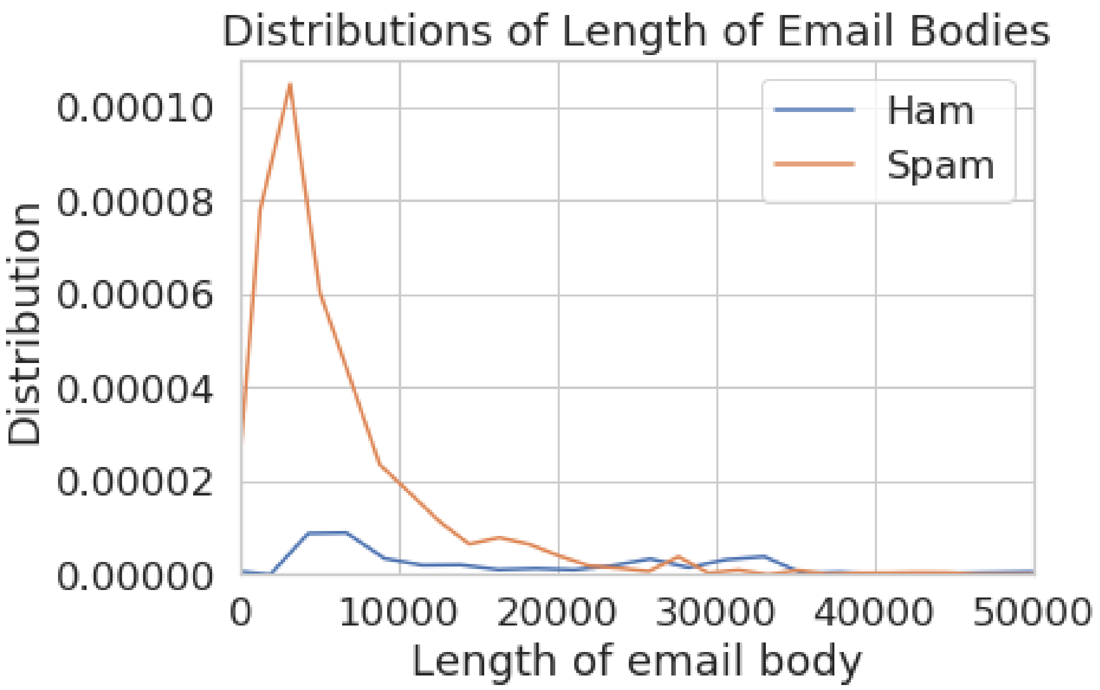
0.0.3 Question 3b

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 [13]: sns.distplot(train[train['spam'] == 0]['email'].apply(len), label='Ham', hist=False)
         sns.distplot(train[train['spam'] == 1]['email'].apply(len), label='Spam', hist=False)

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

```
/srv/conda/envs/data100/lib/python3.6/site-packages/scipy/stats/stats.py:1713: FutureWarning: Using a n
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



0.0.4 Question 6c

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

There will be zero false positives since the zero model predicts no positives whatsoever. Because of this, there will also be 0 true positives because the model never gets it right even if the email is spam. Therefore the recall is 0. There are many false negatives (the amount of false negatives is equal to the amount of spam emails because the model predicts they are ham emails indiscriminately). The accuracy remains high because ~74% of the emails are ham, and since the model predicts all emails are ham, it gets it right around 74% of the time.

0.0.5 Question 6e

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

There are a LOT more false negatives (over 10 times more) than false positives. The model missed a lot of spam.

0.0.6 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. Predicting 0 for every email had around a 74.5% accuracy, so it was fairly close to the logistic regression classifier model.
2. The words used by the logistic regression classifier model likely don't show up much in very many emails, and when they do, it's probably a mix of both spam and ham emails that they show up in.
3. I would prefer the logistic regression model because it has far better recall. The sero model has 0 recall, meaning that it never predicts a true positive. The logistic regression model at least finds some true positives out of the false negatives.

0.0.7 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 picked words that were used very commonly in spam, but very rarely used (if at all) in ham emails.

2. When I tried using words that were not common in either spam or ham (but obviously skewed towards spam like "guarantee" or "money"), I didn't get enough positive hits, so my recall was low. Also when I used co-occuring html tags it was too redudant to do much good, so instead I simply used "html" and assumed all the other tags would co-occur with this word. A single '!' doesn't do much good, but usually only spam uses '!!' 2 exclams or more.

3. "html," "body," "font," and "size" all correspond to mark-up language and are common in spam emails which probably are styled like webpages, but these are not features of spam that immediately come to mind. 'Dear' seems to be commonly used as an opening to spam mail, and references to money or offers also correlate with spam. Using negation like "doesn't," "cannot," "won't" are all uncommon in spam. Past tense is rarely used in spam. Ultimately it was suprising that even though I had a lot of classifier features, I never overfit the data by very much

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

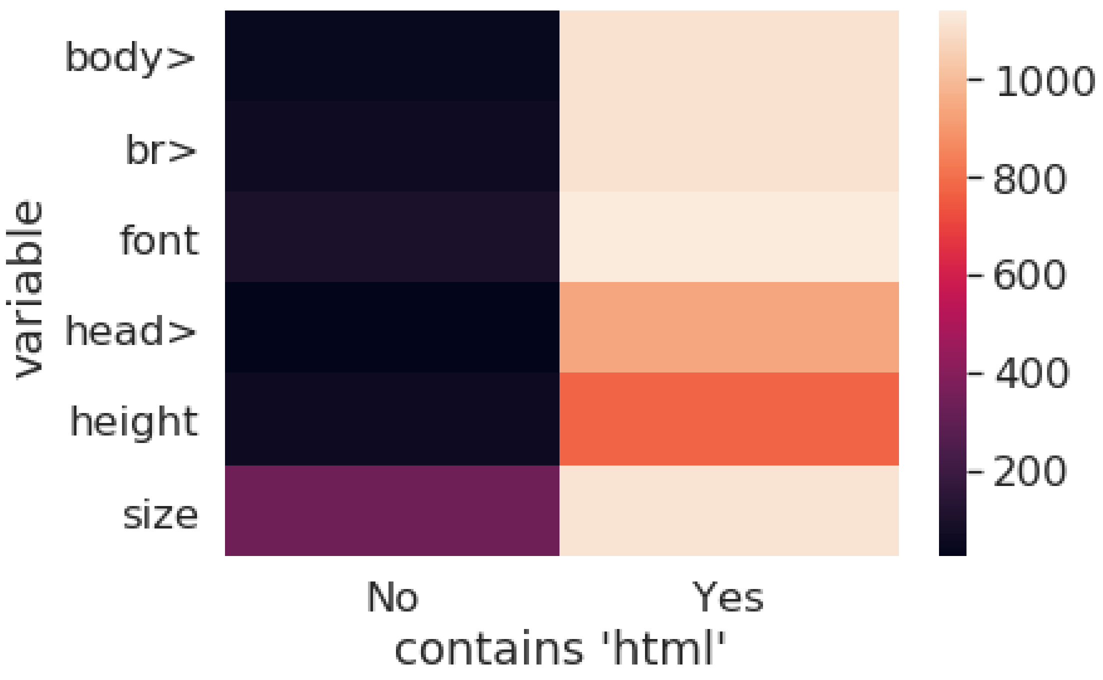
```
In [25]: # Write your description (2-3 sentences) as a comment here:
         # I created a heatmap that charts the frequency in which an email contains '<body>', <br>, 'fo
         # given if that email also did or did not contain the word 'html'.
         # From this we can see that significantly more frequently than not, if an email contains <body
         # 'head', then it also contains 'html', meaning that 'html' can substitute as a feature for al
         # loss.

         # Write the code to generate your visualization here:
         occ = words_in_texts(['font', 'size', 'body>', 'br>', 'head>', 'height'], train['email'])

         df = pd.DataFrame({
             'font': occ.T[0],
             'size': occ.T[1],
             'body>': occ.T[2],
             'br>': occ.T[3],
             'head>': occ.T[4],
             'height': occ.T[5],
             "contains 'html'":  pd.Series(words_in_texts(['<html>'], train['email']).flatten()).replac
         }).melt("contains 'html'")

         pvdf = df.pivot_table(values='value', index=["contains 'html'"], columns='variable', aggfunc=su

         sns.heatmap(pvdf.T);
```



0.0.8 Question 9: 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.

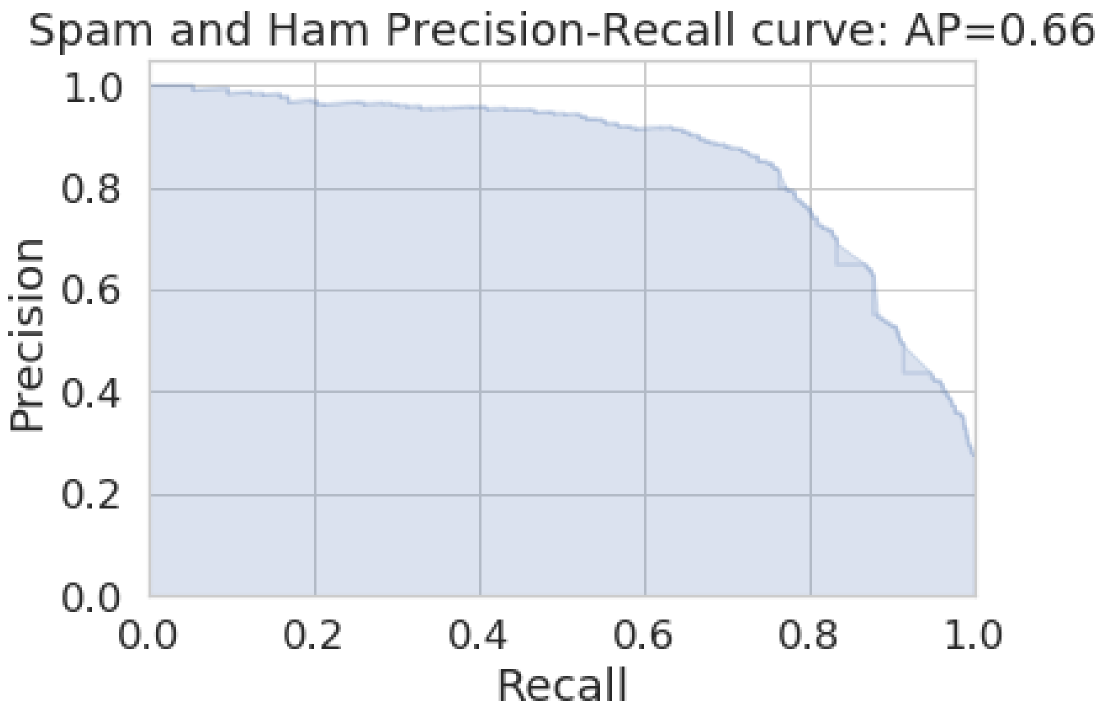
The precision-recall curve shows this trade off for each possible cutoff probability. In the cell below, plot a precision-recall curve for your final classifier (the one you use to make predictions for Kaggle).

```
In [28]: from sklearn.metrics import precision_recall_curve
         from sklearn
         # Note that you'll want to use the .predict_proba(...) method for your classifier
         # instead of .predict(...) so you get probabilities, not classes

         precision, recall, _ = precision_recall_curve(Y_train2, model2.predict_proba(X_train2)[:, 1])

         plt.step(recall, precision, color='b', alpha=0.2,
                  where='post')
         plt.fill_between(recall, precision, alpha=0.2, color='b')

         plt.xlabel('Recall')
         plt.ylabel('Precision')
         plt.ylim([0.0, 1.05])
         plt.xlim([0.0, 1.0])
         plt.title('Spam and Ham Precision-Recall curve: AP={0:0.2f}'.format(np.mean(precision)));
```


Spam and Ham Precision-Recall curve: AP=0.66
Precision
Recall