Notebook

April 27, 2020

0.0.1 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.

Since zero_predictor always predicts 0, it's getting 0 accuracy as it hasn't predict even one true positive; so is recall. Obviously, false positive = 0 and false negative = number of all spams.

0.0.2 Question 6e

Are there more false positives or false negatives when using the logistic regression classifier from Part A?

More false negatives.

0.0.3 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.

1. Only slightly higher (than 74.5%)
2. The classifier can not effectively distinguish spam from ham as words selected are both not so prevalent in ham and spam.
3. I prefer zero_predictor as prediction accuracy is almost the same with logistic regression but I wont miss a single ham.

0.0.4 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:

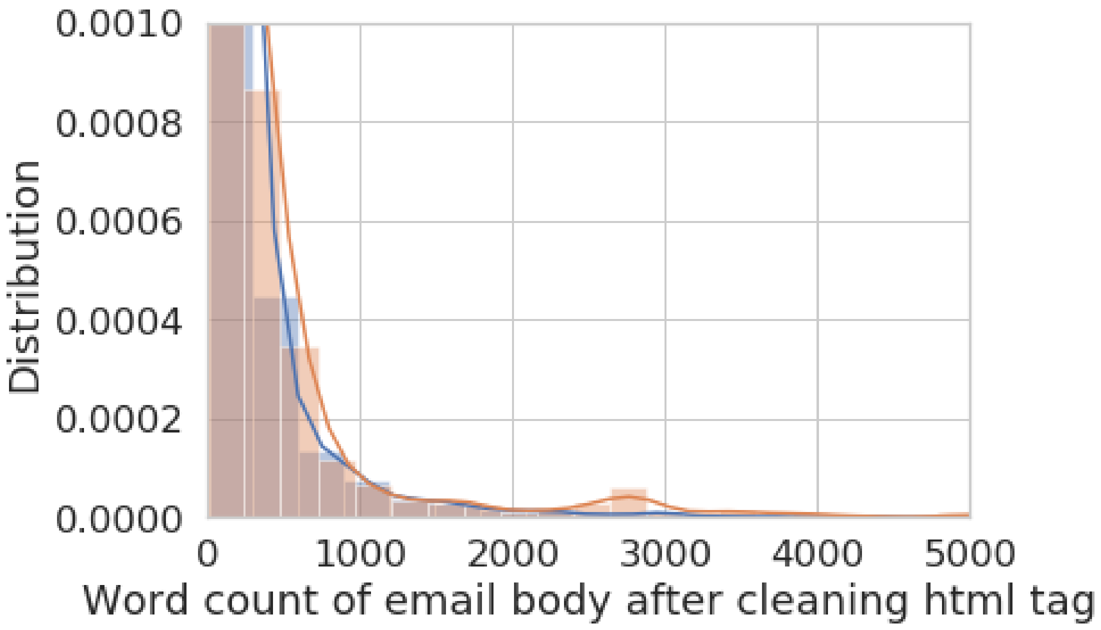
1. How did you find better features for your model?
2. What did you try that worked / didn't work?
3. What was surprising in your search for good features?

1. Visulize data to see if it's good feature in general. Then use sorting rules to prioritize features that distinguish hams from spams obviously and drop others
2. Words count after html tag cleaning is shown below to be not a good feature; neither is "!" count.
3. It's actually surprising to see "!" turned out not a good feature, cause in my personal experience its just so commonly used in spam.

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

```
In [12]: # Write your description (2-3 sentences) as a comment here:
         # According to the plot below, difference between ham and spam distribution is not obvious
         # engough. Because of this words count is selected as a feature.
         #

         # Write the code to generate your visualization here:
         pd.options.mode.chained_assignment = None  # default='warn'
         import re

         ## 1.A length(did before). As we are to do words count, this one is neglected

         ## 1.B Words count. Not selected as a feature.
         def detag(str):
             return re.findall('>([^>]+)<', str)

         def splitIntoWords(str):
             return re.findall('\w+', str)

         def calLength(data):
             html_words_array = ["".join(x) for x in data.loc[data['email'].str.contains("<html>")]['ema
             html_words_series = pd.Series(html_words_array, index = data[data['email'].str.contains("<h

             data['email_detaged'] = data['email']
             data.loc[data['email'].str.contains("<html>"), 'email_detaged'] = html_words_series
             data['email_words']=data['email_detaged'].apply(splitIntoWords)

             data['subject'] = data['subject'].fillna("")
             data['subject_words']=data['subject'].apply(splitIntoWords)

             data['length'] = [np.shape(x)[0] for x in data['email_words']]
             return data

         calLength(train)

         plt.figure()
         sns.distplot(train[train['spam']==0]['length'], hist=True, kde=True, label='Ham')
         sns.distplot(train[train['spam']==1]['length'], hist=True, kde=True, label='Spam')
         plt.xlim(0,5000)
         plt.ylim(0,0.001)
         plt.xlabel("Word count of email body after cleaning html tag")
         plt.ylabel("Distribution");

         ## 1.C "!". Not selected as a feature.
         #plt.figure()
         #sns.distplot(train[train['spam']==0]['email'].str.count('!'), hist=True, label='Ham')
         #sns.distplot(train[train['spam']==1]['email'].str.count('!'), hist=True, label='Spam')
         #plt.xlim(0,100)
         #plt.xlabel("Count of '!'")
         #plt.ylabel("Distribution");

         ## 1.D Capital letters won't work since training data is processed into lower-cased form. Not
## 1.E Re or Fwd. Not selected as features alone, but included in subject_words
train['subject'] = train['subject'].str.lower()
special_sbj = ['re:','fwd:']

#print(train.isnull().sum())
train=train.fillna("")

ham_count = np.sum(words_in_texts(special_sbj, train[train['spam']==0]['subject']),axis=0)
ham_proportion = ham_count/train[train['spam']==0].shape[0]
spam_count = np.sum(words_in_texts(special_sbj, train[train['spam']==1]['subject']),axis=0)
spam_proportion = spam_count/train[train['spam']==1].shape[0]
plot_data = pd.DataFrame([ham_proportion,spam_proportion], columns=special_sbj, index = ['ham'
plot_data = plot_data.stack()
plot_data = plot_data.reset_index()
plot_data.columns = ['Type','Words','Proportion of Emails']
#plt.figure()
#sns.barplot(data=plot_data, hue = 'Type', x = 'Words', y = 'Proportion of Emails');

## 2 Words Selection. selected_words as features.

from collections import Counter

ham_words = np.concatenate(train[train['spam']==0]['email_words'].sort_index(axis=0, ascending
spam_words = np.concatenate(train[train['spam']==1]['email_words'].sort_index(axis=0, ascendin

ham_words_counter = Counter(ham_words)
spam_words_counter = Counter(spam_words)

ham_words_counter_df = pd.DataFrame.from_dict(ham_words_counter, orient='index')
spam_words_counter_df = pd.DataFrame.from_dict(spam_words_counter, orient='index')

proportion_words_df = (spam_words_counter_df/ham_words_counter_df).rename(columns={0: "proport
ham_words_counter_df = ham_words_counter_df.rename(columns={0: "ham"})
spam_words_counter_df = spam_words_counter_df.rename(columns={0: "spam"})

words_count = ham_words_counter_df.join(spam_words_counter_df).join(proportion_words_df)
words_count = words_count.sort_values(by="proportion", ascending=False)


ham_subjects = np.concatenate(train[train['spam']==0]['subject_words'].sort_index(axis=0, ascen
spam_subjects = np.concatenate(train[train['spam']==1]['subject_words'].sort_index(axis=0, asc

ham_subjects_counter = Counter(ham_subjects)
spam_subjects_counter = Counter(spam_subjects)

ham_subjects_counter_df = pd.DataFrame.from_dict(ham_subjects_counter, orient='index')
spam_subjects_counter_df = pd.DataFrame.from_dict(spam_subjects_counter, orient='index')

proportion_subjects_df = (spam_subjects_counter_df/ham_subjects_counter_df).rename(columns={0:
ham_subjects_counter_df = ham_subjects_counter_df.rename(columns={0: "ham"})
spam_subjects_counter_df = spam_subjects_counter_df.rename(columns={0: "spam"})

subjects_count = ham_subjects_counter_df.join(spam_subjects_counter_df).join(proportion_subjec
subjects_count = subjects_count.sort_values(by="proportion", ascending=False)
```

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


Distribution
Word count of email body after cleaning html tag