
0.1 Question 1

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

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

I found better features for my model by experimenting with the common features in spam emails. I did this by playing around with different words (hence my long list of words in spam emails) as well as incorporating the suggestions from the question. I ended up starting off by including every single suggsted feature from the question and then worked my down from there, eventually take out the one hot encoding and the number of characters in the subject line and the body of the email.

I started off by trying every single feature separately, then pairing them up, then grouping them. This allowed me to see which combinations of features worked best. I found that it sometimes worked, but I also found that oftentimes I was putting in too much (like I tried to use onehotencoder and the word features and the subject line character length and punctuation (etc) and this actually made my model less accurate so I went back and limited those features.

In my search for good features, I was surprised that some of the feature words that I wanted to include like 'free cash' and 'free money' actually lowered my accuracy. I also tried to use all of the suggested features given in the question, and found that for the character length of the subject line and character length of the email, my accuracy was actually lowered (probably because this was not linearly independent from the word length of the email and subject). This is probably because these are probably not as indicative of the type of email (spam or ham).

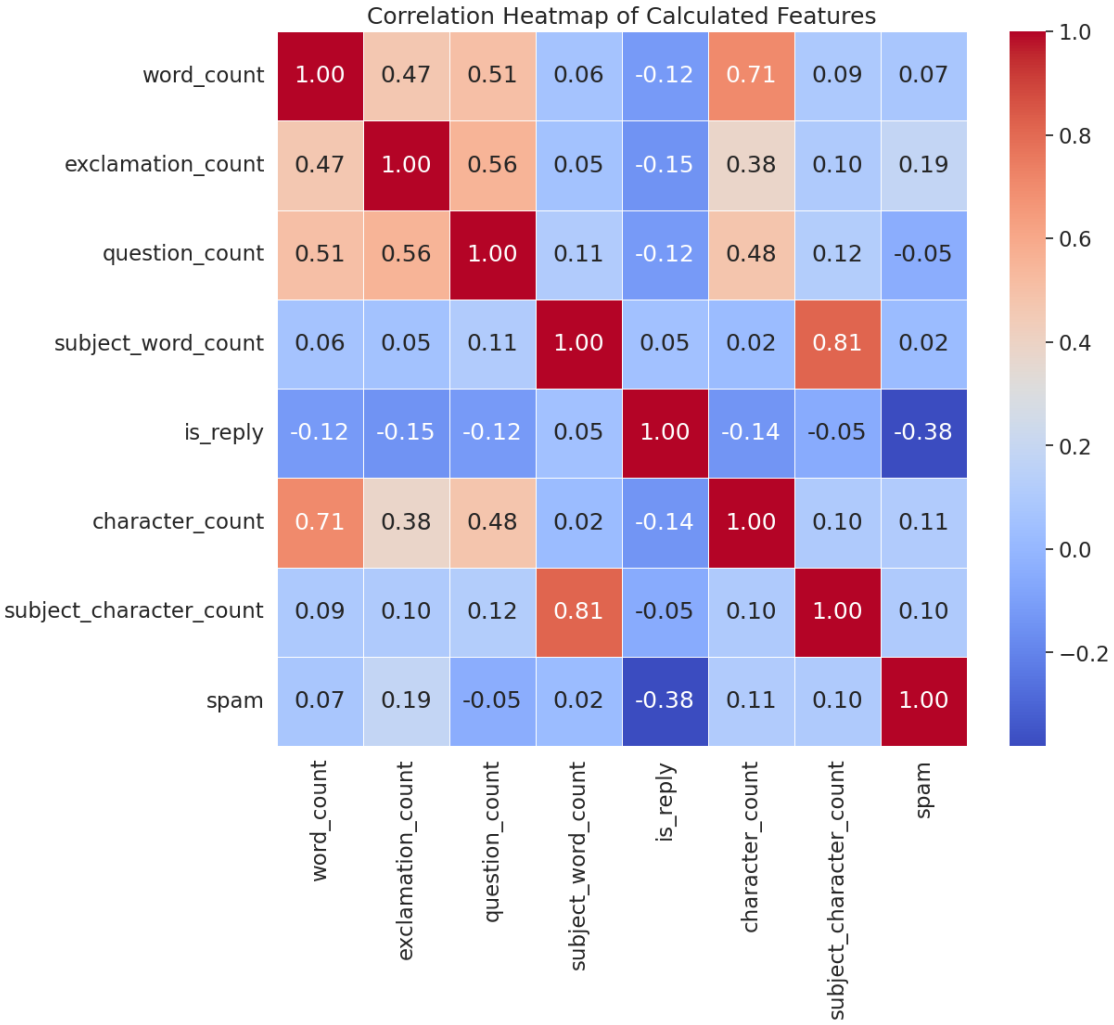
0.2 Question 2a

Generate your visualization in the cell below.

In [90]:
```
train_copy = train.copy()
train_copy['word_count'] = train_copy['email'].apply(lambda x: len(x.split()))
train_copy['exclamation_count'] = train_copy['email'].apply(lambda x: x.count('!'))
train_copy['question_count'] = train_copy['email'].apply(lambda x: x.count('?'))
train_copy['subject_word_count'] = train_copy['subject'].apply(lambda x: len(x.split()))
train_copy['is_reply'] = train_copy['subject'].apply(lambda x: 1 if 'Re:' in x else 0)
train_copy['character_count'] = train_copy['email'].apply(lambda x: len(x))
train_copy['subject_character_count'] = train_copy['subject'].apply(lambda x: len(x))

feature_correlation = train_copy[['word_count', 'exclamation_count', 'question_count',
                                  'subject_word_count', 'is_reply', 'character_count',
                                  'subject_character_count','spam']].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(feature_correlation, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap of Calculated Features')
plt.show()
```


Correlation Heatmap of Calculated Features
word_count 1.00 0.47 0.51 0.06 -0.12 0.71 0.09 0.07
exclamation_count 0.47 1.00 0.56 0.05 -0.15 0.38 0.10 0.19
question_count 0.51 0.56 1.00 0.11 -0.12 0.48 0.12 -0.05
subject_word_count 0.06 0.05 0.11 1.00 0.05 0.02 0.81 0.02
is_reply -0.12 -0.15 -0.12 0.05 1.00 -0.14 -0.05 -0.38
character_count 0.71 0.38 0.48 0.02 -0.14 1.00 0.10 0.11
subject_character_count 0.09 0.10 0.12 0.81 -0.05 0.10 1.00 0.10
spam 0.07 0.19 -0.05 0.02 -0.38 0.11 0.10 1.00
word_count
exclamation_count
question_count
subject_word_count
is_reply
character_count
subject_character_count
spam
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

0.3 Question 2b

Write your commentary in the cell below.

This heatmap is showing the correlations between different features I was testing for and the 'spam' variable which is our target variable in the email dataset. Every cell is the correlation coefficient between the corresponding features, and this correlation matrix showed me that spam is pretty correlated to if the email is a reply and also exclamation marks. I was surprised that it has low correlations with the word count for both the body and the subject, and I took this into account when outlining features. I also saw potential multicollinearity between word count and character count (which makes sense) so I decided to only include one or the other in my features for Question 1.

0.4 Question 3: ROC Curve

In most cases, we won't be able to get 0 false positives and 0 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 that they have cancer until it's too late. In contrast, a patient can receive another screening for a false positive.

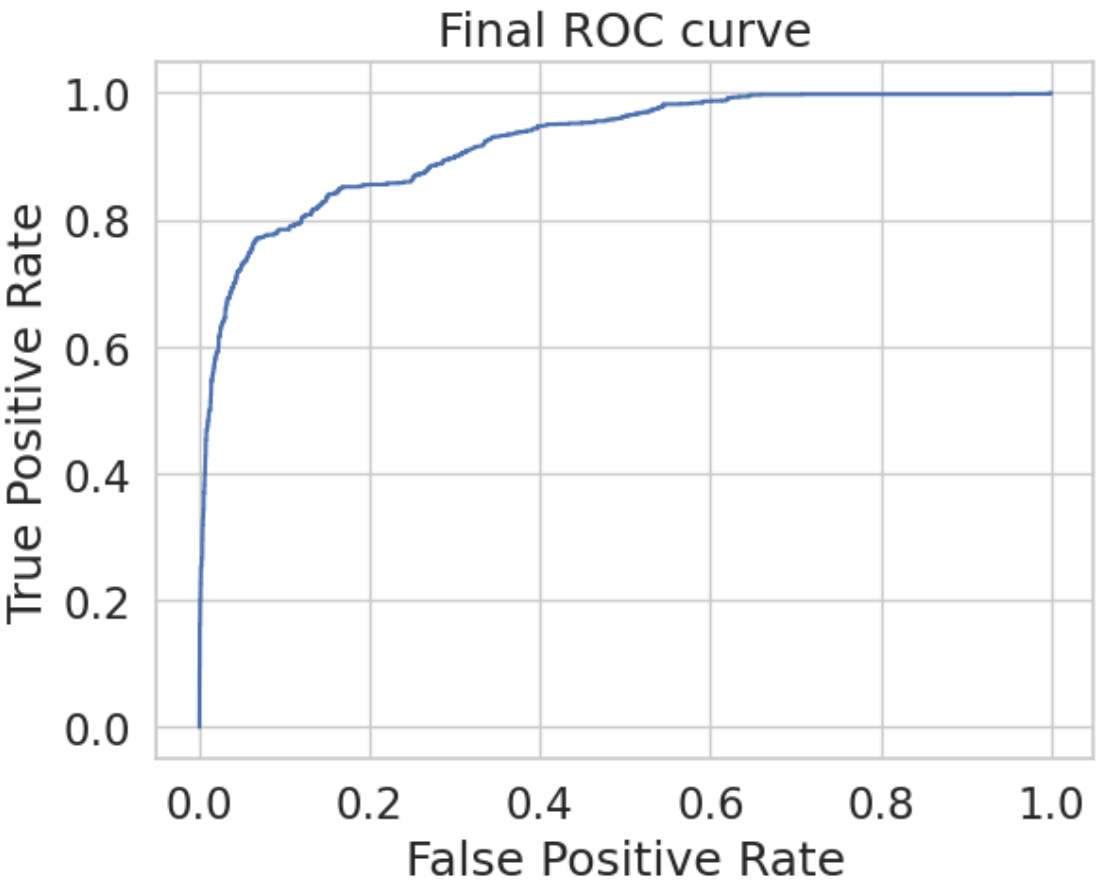
Recall that logistic regression calculates the probability that an example belongs to a particular class. To classify an example, we say that an email is spam if our classifier gives it ≥ 0.5 probability of being spam. However, **we can adjust that cutoff threshold**: We can say that an email is spam 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 Receiver Operating Characteristic (ROC) curve shows this trade-off for each possible cutoff probability. In the cell below, plot an ROC curve for your final classifier (the one you use to make predictions for Gradescope) on the training data. Refer to Lecture 23 to see how to plot an ROC curve.

Hint: You'll want to use the `.predict_proba` method for your classifier instead of `.predict` to get probabilities instead of binary predictions.

```
In [91]: X_train_ROC = extract_features(train, some_words)
         Y_train = np.array(train['spam'])
         my_model = LogisticRegression(solver = 'lbfgs', max_iter = 1000)
         my_model.fit(X_train_ROC, Y_train)
         print('Accuracy:', my_model.score(X_train_ROC, Y_train))
         Y_predict = my_model.predict_proba(X_train_ROC)[:, 1]
         fpr, tpr, thresholds = roc_curve(Y_train, Y_predict)
         #with sns.axes_style("white"):
         plt.plot(fpr, tpr)
         plt.title("Final ROC curve")
         plt.xlabel("False Positive Rate")
         plt.ylabel("True Positive Rate")
         #plt.xlim([0, 1])
         #plt.ylim([0, 1])
         plt.show()
```

```
Accuracy: 0.8925861839478237
```


Final ROC curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0