
0.1 Question 1

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 or didn't work?
3. What was surprising in your search for good features?

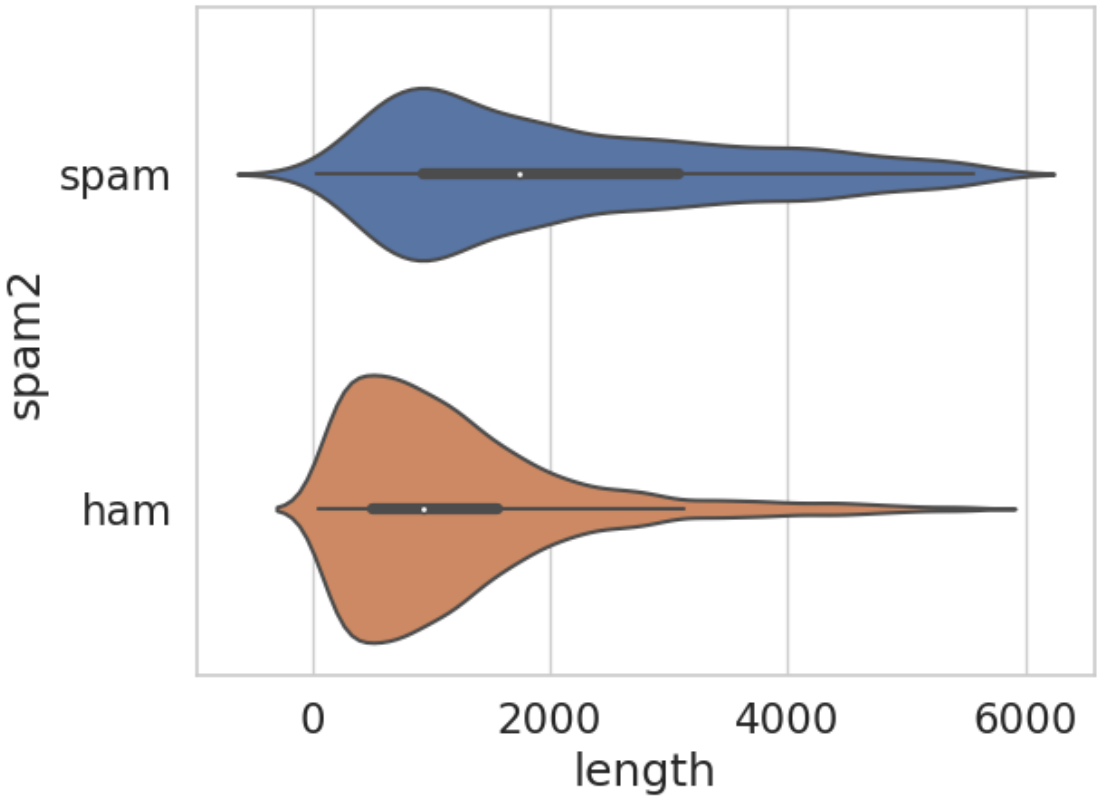
1. I found better features by comparing the ratios of punctuantions and certain words in spam versus "ham" emails to find which had the greatest differnce in an attempt to further seperate them with data. Also I thought a good feature would be length of either the subject or body. I compared the two and found I got higher accuracy when using just the length of the body.
2. Something I tried that didn't work was implementing a function that compared upper to lower case ration in emails. This just didn't work because I couldnt figure out a fucntion to extrac t this data from the orignal data. I think thus would have been very powerful and helped a lot in increasing accuracy of my model.
3. Something suprising was the power of choosing the right words to compare and finding words with the biggest differnce between ham and spam. So once I realized how powerful this was I looked at punctuation and this is what really pushed my model over the 85% accuracy marker.

0.2 Question 2a

Generate your visualization in the cell below.

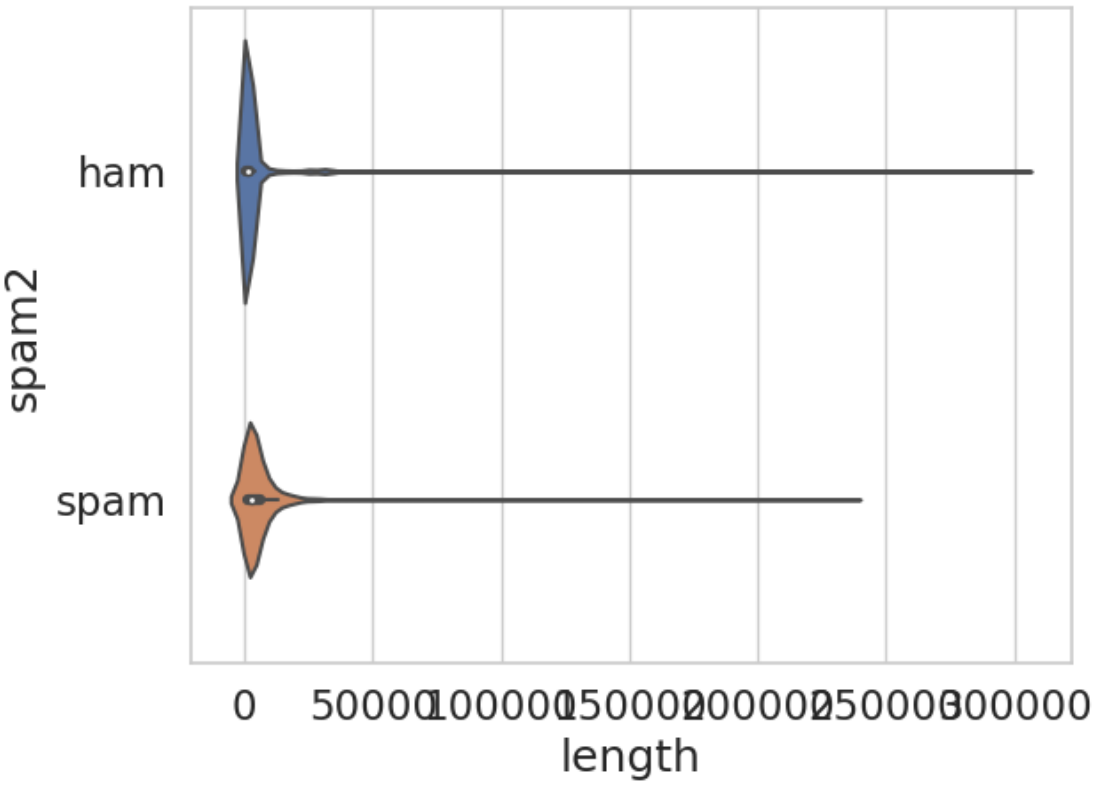
In [61]:
```
def helper():
    list = []
    for i in train['spam']:
        if i == 0:
            list.append('ham')
        else:
            list.append('spam')
    return list
train['length'] = body_length(train)
train['spam2'] = helper()
train = train.sort_values('length', ascending = False)
train2 = train[1000:]
sns.violinplot(data = train2, x='length', y='spam2')
```

Out[61]:
```
<AxesSubplot:xlabel='length', ylabel='spam2'>
```



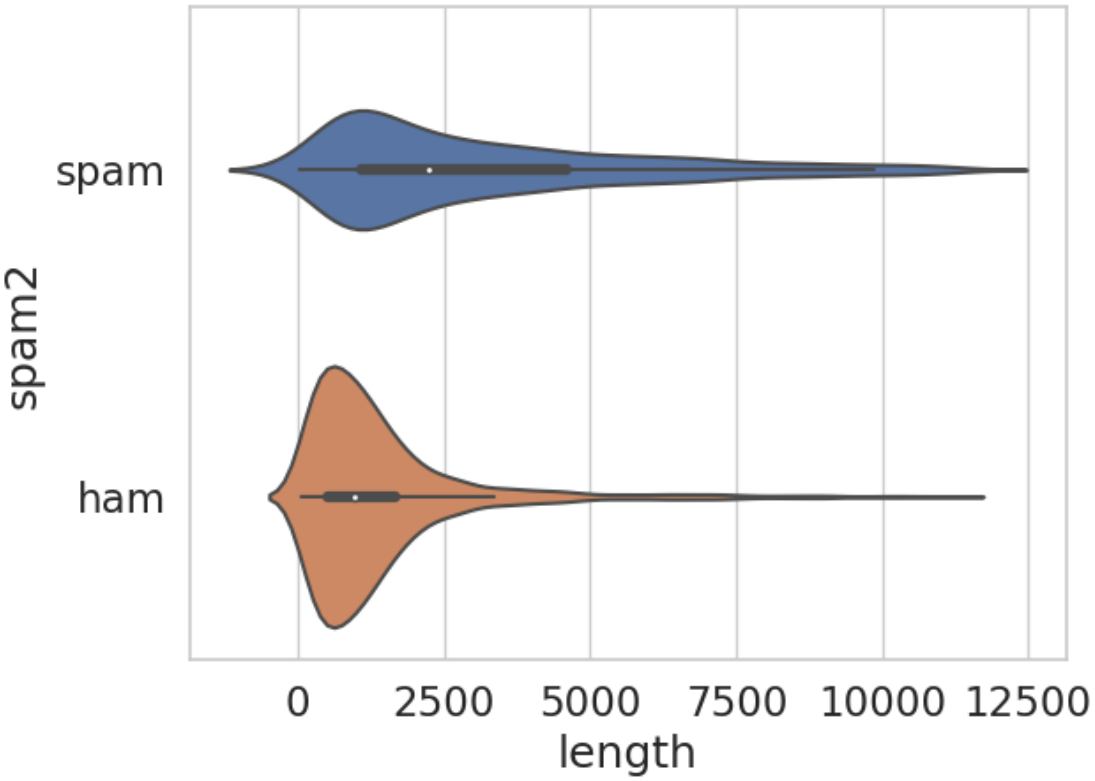

```
In [62]: sns.violinplot(data = train, x='length', y='spam2')
```

```
Out[62]: <AxesSubplot:xlabel='length', ylabel='spam2'>
```



In [63]: sns.violinplot(data = train[500:], x='length', y='spam2')

Out[63]: <AxesSubplot:xlabel='length', ylabel='spam2'>


spam
ham
spam2
0
2500
5000
7500
10000
12500
length

0.3 Question 2b

Write your commentary in the cell below.

Aside from the plots aobve I also looked at bar plots comapring the ratios of words to total nubmer of words. I found that the spam emails had far less of super common words like (the, a, and, of, you) A.K.A filler words which are common in sentences between conversation. So usiong intuition this makes sense. Then I ended up lookig at the length of the body of the emails to comapre spam versus ham. For my chart I choose a violin plot because I thought it would be the easiest to see the differnce in the groups visually. From my plots we can see that spam emails on average tend to be longer but ham emails will reach lengths far greater than spam emails ocassionly. Which all makes sense intuitively. Also we can see that usually the spam emails are all about the same lenght, but often still long due to the random characters and links would be my best guess.

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, whereas a patient can just receive another screening for a false positive.

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 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 a ROC curve for your final classifier (the one you use to make predictions for Gradescope) on the training data. Refer to Lecture 24 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 [65]: import sklearn
         import sklearn.linear_model
         import plotly.offline as py
         import plotly.express as px
         X_ROC = process_data(train)
         Y_ROC = train['spam']
         linear_regression = sklearn.linear_model.LogisticRegression(fit_intercept=True)
         linear_regression.fit(X_ROC,Y_ROC)
         precision, recall, threshold = precision_recall_curve(Y_ROC, linear_regression.predict_proba(X_
         fig = px.line(x=recall[:-1], y=precision[:-1], hover_name=threshold)
         fig.update_xaxes(title="Recall")
         fig.update_yaxes(title="Precision")
```