0.0.1 Question 1: 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 or didn't work?
3. What was surprising in your search for good features?

One way I improved my model was by choosing better attributes to classify a spam email. Since a large part of my model relies on counting the ocurrences of specific characters/words in the subject/email, it is very important to choose words that are good indicators of spam. Therefore, I wrote a function to calculate the ratio between the number of occurences of the word in spam vs ham, each normalized by the number of spam/ham. Therefore, if the ratio is 10, then we can say the word shows up 10x more in spam than ham. I then visually skimmed through spams and used my intuition to choose many candidates. I then tested them using the function I created and selected the ones that have extremely high or low ratios, since they should be good indicators. This worked very well and my model's accuracy immediately increased to around 90% after adding around 30 words. It was surprising for me to find words that I previously expected to be in spams to actually be more prevalent in hams, such as "news".

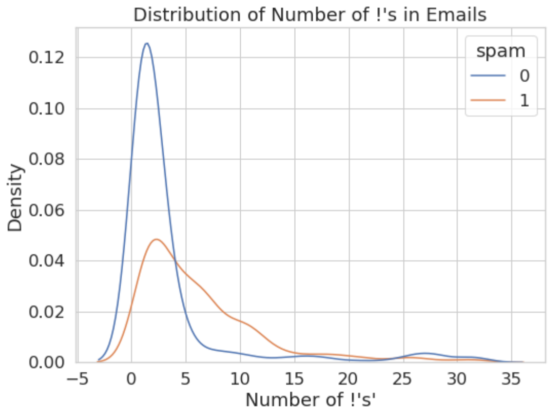
Question 2a Generate your visualization in the cell below.

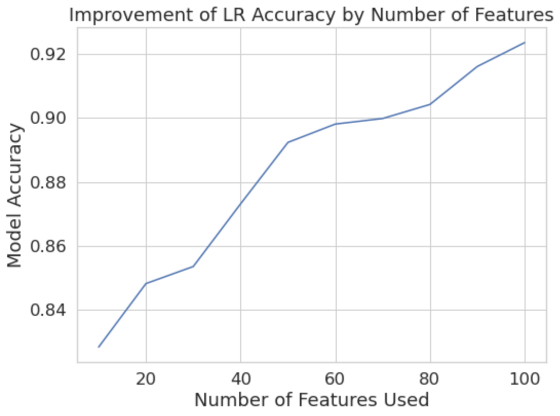
```
In [128]: fig, axes = plt.subplots(1, 2, figsize = (18, 6))

          #histogram of exclaimation marks
          train_copy = train.copy()
          train_copy["number of !'s"] = train_copy["email"].apply(lambda x: str(x).count("!"))
          #remove outliers
          train_copy = train_copy[(train_copy["number of !'s"] > 0) &
                                  (train_copy["number of !'s"] <= np.percentile(train_copy["number of !
          sns.kdeplot(data = train_copy, x = "number of !'s", hue = "spam", ax = axes[0])
          axes[0].set_title("Distribution of Number of !'s in Emails")
          axes[0].set_xlabel("Number of !'s'")

          #line plot of accuracy improvement by number of words used
          words = ["enenkio", "mortgage", "mailings", "islands", "secrets", "grants", "insurance", "war
               "guaranteed", "adult", "professionals", "removal", "business", "only", "valuable", "loan
               "absolutely", "marketing", "membership", "cash", "thousands", "payment", "replying", "re
               "guarantee", "huge", "risk", "within", "card", "interest", "offers", "opportunity", "rea
               "free", "please", "new", "money", "receive", "send", "click", "want", "removed", "best",
               "need", "credit", "million", "site", "offer", "special", "online", "link", "pay", "acces
               "simply", "phone", "easy", "guide", "dollars", "simple", "product", "legal", "income", "
               "geneva", "help", "transaction", "confidential", "adult", "limited", "sex", "respond", "
               "software", "sites", "spy", "profit", "profits", "provide", "for you", "congratulations"
               "find out", "html", "<", "mom", "dad", "proven", "sexy", "girls", "horny", "excit", "new
               "regarding", "instructions"]
          def get_word_counts_2(data, sample_size):
              data = data.copy().reset_index()
              words_sampled = words[:sample_size]
              data = pd.merge(
                  left = data, right = pd.DataFrame(words_in_texts(words_sampled, data["email"]), colum
                  left_index = True, right_index = True
              )
              return data

          num_words_used = range(10, len(words), 10)
          accuracy_by_num_words_used = []
          for num_words in num_words_used:
              model_2 = LogisticRegression(fit_intercept = True, solver = 'lbfgs', max_iter = 100)
              model_2.fit(X = get_word_counts_2(train, num_words).iloc[:, 5:], y = train["spam"])
              accuracy = model_2.score(X = get_word_counts_2(train, num_words).iloc[:, 5:], y = train["
              accuracy_by_num_words_used.append(accuracy)
          sns.lineplot(num_words_used , accuracy_by_num_words_used, ax = axes[1])
          axes[1].set_title("Improvement of LR Accuracy by Number of Features")
          axes[1].set_xlabel("Number of Features Used")
          axes[1].set_ylabel("Model Accuracy");
```


Distribution of Number of !'s in Emails
Density
Number of !'s'
spam
0
1
0.00
0.02
0.04
0.06
0.08
0.10
0.12
−5
0
5
10
15
20
25
30
35


Improvement of LR Accuracy by Number of Features
Model Accuracy
Number of Features Used
0.84
0.86
0.88
0.90
0.92
20
40
60
80
100

Question 2b Write your commentary in the cell below.

As we can see from the plot on the left, which compares the distributions of the number of exclaimation marks in spams vs hams, spams appears to have more exclaimation marks than hams. We can make this conclusion since the distribution of spams have a higher mean and is less skewed to the right. This makes sense since we would expect the tone of spam emails to be more uplifting in order to prompt you to click its link or follow its instructions. As a result of this EDA, we find that it is promising to use the number of exclaimation marks as a feature.

As we can see from the plot on the right, which plots the accuracy of a model by its number of features (the number of words checked for occurence in this case), as the number of features increases, the accuracy of the model increases as well. This is not surprising, since more features naturally improves model bias. However, it is important to see at what model complexity do we pass the 88% mark. Moreover, it is interesting to see how the rate of improvement changes, which in this case, appears to be concave, meaning that the rate of improvement is decreasing, which also intuitively makes sense.

0.0.2 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.

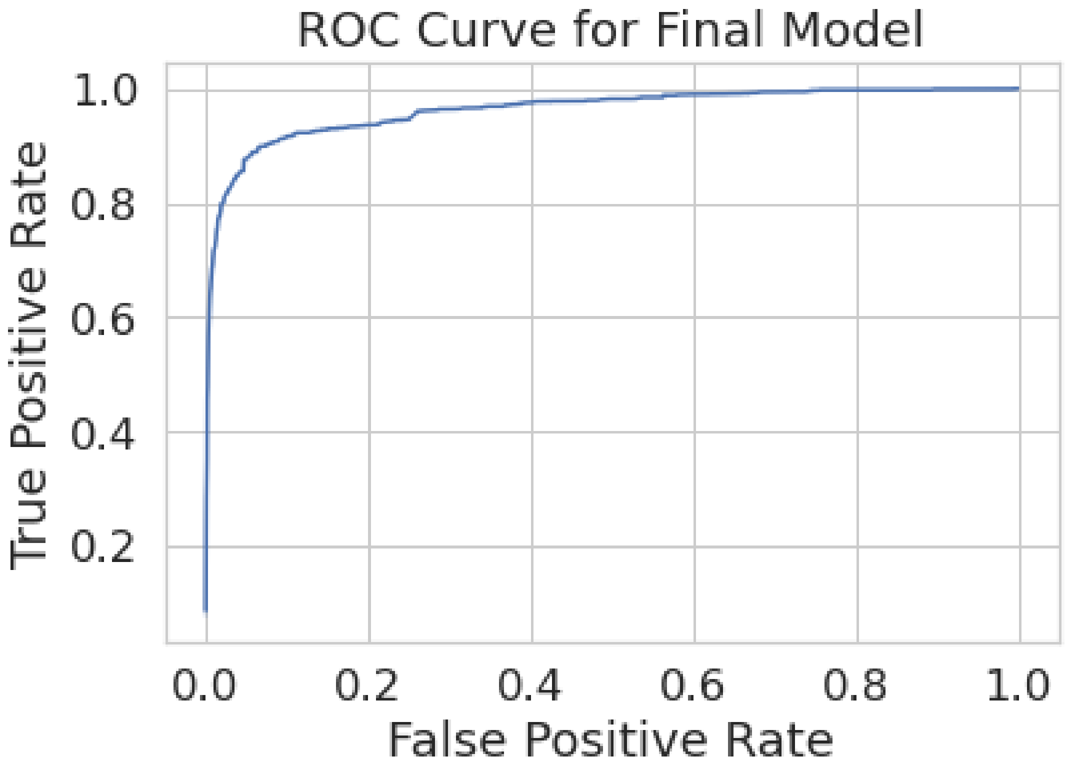
Recall that logistic regression calculates the probability that an example belongs to a certain class. Then, to classify an example we say that an email is spam if our classifier gives it ≥ 0.5 probability of being spam. However, *we can adjust that cutoff*: we can say that an email is spam only if our classifier gives it ≥ 0.7 probability of being spam, for example. This is how we can trade off false positives and false negatives.

The 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 20 to see how to plot an ROC curve.

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

```
In [152]: from sklearn.metrics import roc_curve

          fprs, tprs, thresholds = roc_curve(Y_train, model.predict_proba(X_train)[:, 1])
          sns.lineplot(fprs, tprs)
          plt.title("ROC Curve for Final Model")
          plt.xlabel("False Positive Rate")
          plt.ylabel("True Positive Rate");
```


ROC Curve for Final Model
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0