# Study 1: Email Spam Detection

## Approach

### Exploratory Data Analysis

As an initial exploration, we inspected the data as a pandas dataframe and observed the zeros and missing data appeared to be evenly distributed. This was supplemented with a count, finding that each feature had roughly 300 missing entries, and that zero entries accounted for 82% of the non missing data. Considering this, we believed that a constant zero imputation with min-max scaling would work best. To verify this, we ran some logistic regression models with the following results.

True Positive Rate at False Positive Rate = 0.01 for Logistic Regression with C = $10^5$

| | Mean Imputation | Constant 0 Imputation |
|---|---|---|
| Min-Max Scaling | 0.293 | 0.256 |
| No Scale | 0.305 | 0.318 |

We drew a few conclusions:

1. The data is likely non-linear.
2. Min-max scaling does not work with logistic regression.
3. Constant imputation may not work. Certain words like "Hello" may appear at once, are common, and also have some impact on the label, so using a mean imputation method is safer. A KNN imputation was not considered as there were too many features and missing values accounted for 20% of the data.

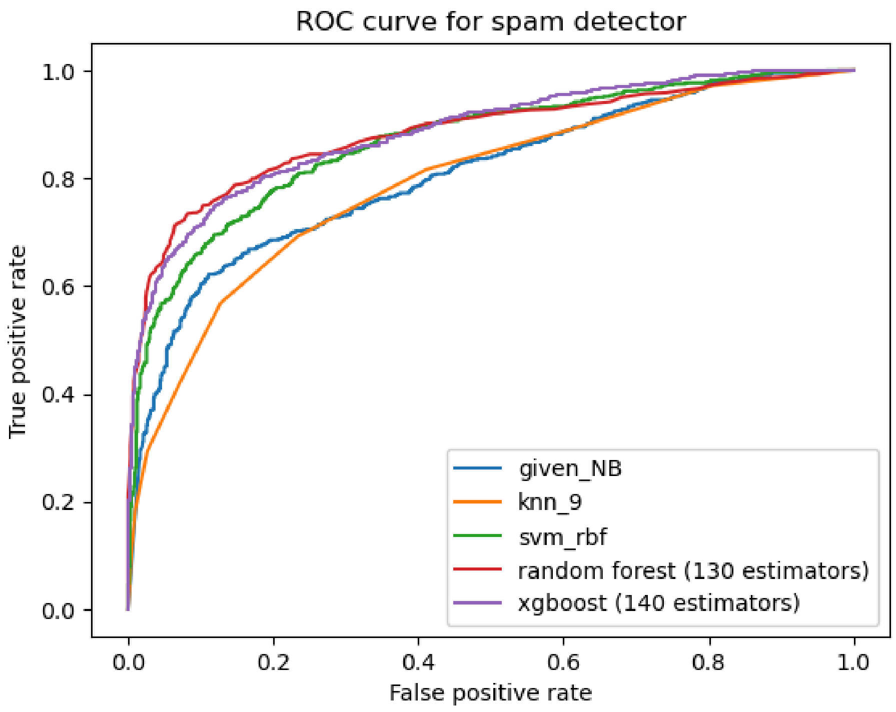
To analyze the features' impact on the label, we computed the correlation heat map between the feature and the label, which displayed values between -0.05 and 0.05, suggesting no correlation. We also calculated the mean of each feature by label and the difference between the means to see if there were features that contributed strongly to the label, but similarly, no results were found. We graphed a two-dimensional PCA, t-SNE, and LDA, and as expected, we did not discover a breaking point. We did think about running a script on personal emails that tracked 30 random words to try to understand the values of the features more but decided against it as the potential gain may not be realized given the deadline.

At this stage, we decided to run some tests on some select models. The naive Bayes' assumption didn't sit well with us. Without knowing which words the features represented, article words such as "the" may appear often. In general, the conditional independence assumption of the words felt problematic without knowing what words the features represented. We chose not to test decision trees in favor of a random forest. We ruled out models that assumed linear separability as we believed the data would be non-linear due to the logistic regression experiments. We conducted tests on Kernel SVM, KNN, Random Forest Classifier, and XGBoost.

Testing

While it was time-consuming, we determined through experimentation that mean imputation resulted in the best results. This may relate to the possibility that frequent words such as "deal" may appear more frequently in spam emails, which increases the model's accuracy. Constant imputation using 0 suggests that the word didn't appear at all in the email, which is very possible considering that zeros account for the majority of the data. However, using constant imputation would increase the zero entries to 85% of the data.

We wrote a testing_classifiers script that evaluated each model using grid search and compiled the best classifiers in the graph below. Holdout validation was used with the training set being spamTrain1 and the test set being spamTrain2.



From the graph, KNN likely did not work as there were too many features. At the same time, implementing feature extraction to further the KNN model seemed to be risky as the words are unknown. XGBoost and Random Forest outperformed the other models, so we decided to focus on them.

## Algorithm

After testing and comparing random forest and boosting, we found that boosting performs slightly better than random forest. The biggest reason we chose boosting is because we believe the data has a higher bias because most of the data consists of zeros. Since boosting reduces bias and random forest reduces variance, we chose to further the boosting algorithm.

In choosing XGBoost over other boosting implementations, the time consumption due to the serial nature of boosting is mitigated as XGBoost uses multiple cores. Boosting is typically vulnerable to outlier data, but after reviewing the mean and standard deviation of each label for each feature, no mean or standard deviation appeared distinctly different from others. As a result, while there may exist outlier data, they are rare enough to have little impact on the model.

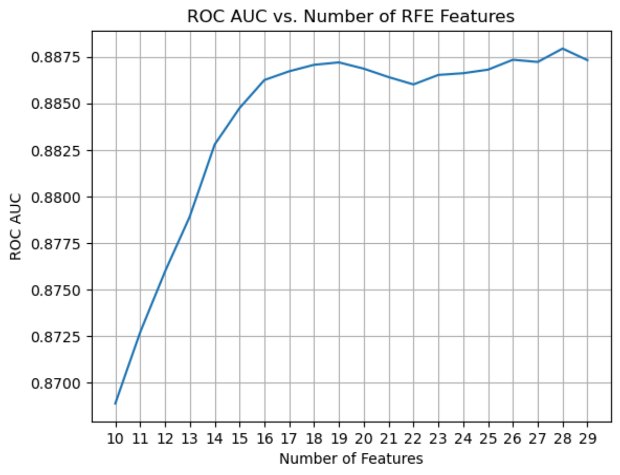
### Preprocessing

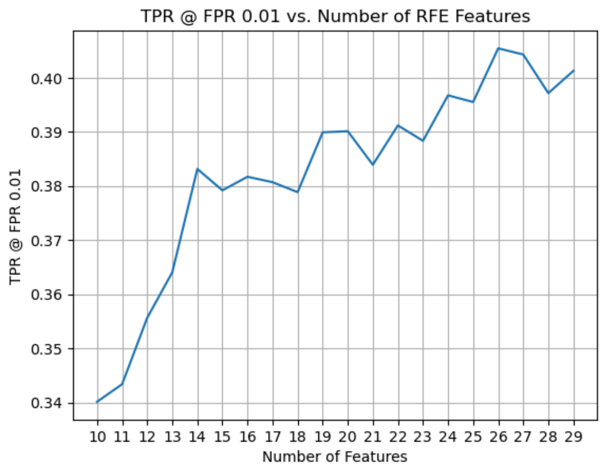
Learning from the exploratory data analysis, we avoided constant imputation as the logistic regression model behaved randomly and chose mean imputation instead. One reason for this is that a missing value does not imply that the word did not appear in the email. A better estimation for the value is to base it on how often the word appears in other emails, which is why we suspect mean imputation obtained higher accuracies than constant imputation.

For scaling, we avoided min-max similarly because the results from our exploratory data analysis did not go well. A standard scaler works better because it better accounts for the zeros in the data. In other words, min-max scaling will leave values originally at zero alone and scale the non-zero values. On the other hand, the standard scaler will consider all the zeros in the feature vector and account for them in the scaled feature vector. We believe this scaling method gave better results because it captured the impact of the zero values on the class label.

### Feature Elimination

Our expectations for testing feature elimination were low, but we decided that there may be a possibility that a word might have little impact on the label. Our attempt at sequential backward selection displayed poor results. Below are the results for recursive feature elimination:



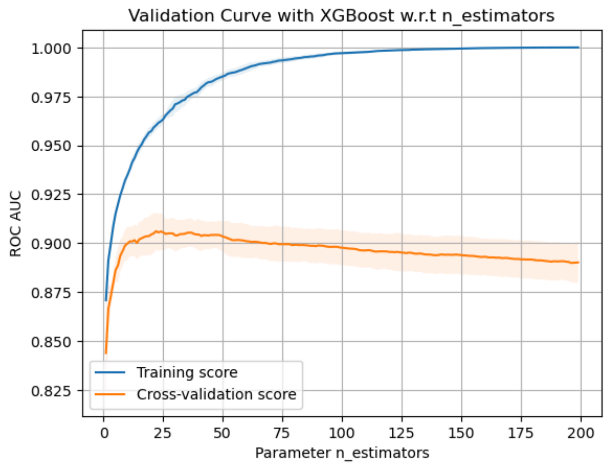


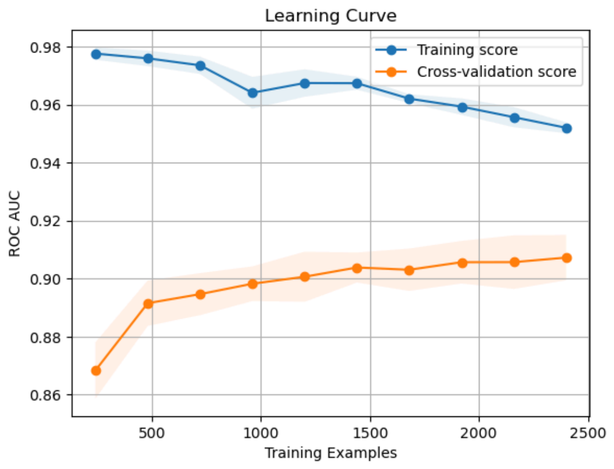
From the graphs, keeping 16 to 18 features would prevent overfitting and lead to a stabler result, but as our goal was to maximize both the AUC and True Positive Rate at False Positive Rate = 0.01, we decided that 26 features is the optimal number of features to keep.

At this point, the hyperparameters of our algorithm use log-loss as the evaluation metric, as the problem is a binary classification. Since our model accuracy wasn't improving as much as we'd like, we attempted a dense neural network to check if perhaps a more complex model would better fit the data. The accuracy hovered around 60% and we quickly returned to our original algorithm.

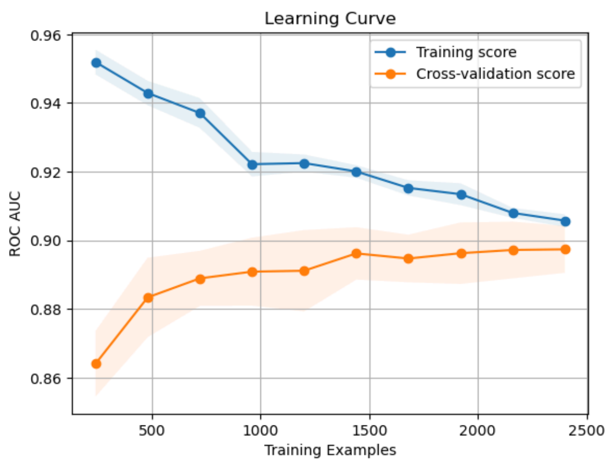
## Hyperparameter Tuning

To further improve our model we turned to 5-fold validation curves and learning curves for hyperparameter tuning. Below are the results.





Evidently, 140 estimators are not necessary. We chose 15 estimators and continued on to arrive at this result:



n_estimators=15, gamma=1, max_depth=3, eval_metric='logloss', random_state=1, alpha=0.1

Our final choice using XGBoost with the above hyperparameters, with mean imputation and standard scaling. We attempted another RFE that eliminated 15 features, surprising us greatly (See Figure 5). We suspect the data may have been engineered so that exactly half the features were noise. The final classifier gave a test set AUC: 0.878 and TPR at FPR = 0.01: 0.311.

## Recommendation

For a personalized spam filter, a confusion matrix can be used to evaluate the algorithm. The goal should be to minimize false positives because we believe that the user is unlikely to see spam emails, and if an important email is incorrectly classified as spam, they may never see it again. The second part would be to maximize the true positives. Combining these two criteria, an F1 score that measures precision (false positives) and recall (true positives) can be used as an evaluation metric. According to our test, our model fails to offer the desired effectiveness, as the true positive rate is .34 at a false positive rate of 0.01. This means that out of 100 emails, if one legitimate email is to be marked as spam, 31 spam emails will likely be marked as spam, and the other emails may be marked correctly as not spam or incorrectly as not spam. In this case, marking emails incorrectly as not spam is not necessarily harmful. If this value is too high, then the spam filter will not work effectively at filtering out spam emails.

One method to solicit feedback from users is to allow them to mark emails as "Not Spam" when they are mistakenly flagged as spam and "Spam" when spam emails bypass the filter. This can provide more data for retraining the model. Surveys can be done to gather data about whether too many legitimate emails are being flagged or if too much spam is getting through, which could change the false positive rate value we try to maximize. Another option is to use more relevant keywords as the features, in which case a new approach may be needed. We could also conduct further experiments on the selected 15 features, approaching it as a separate dataset.

## Appendix

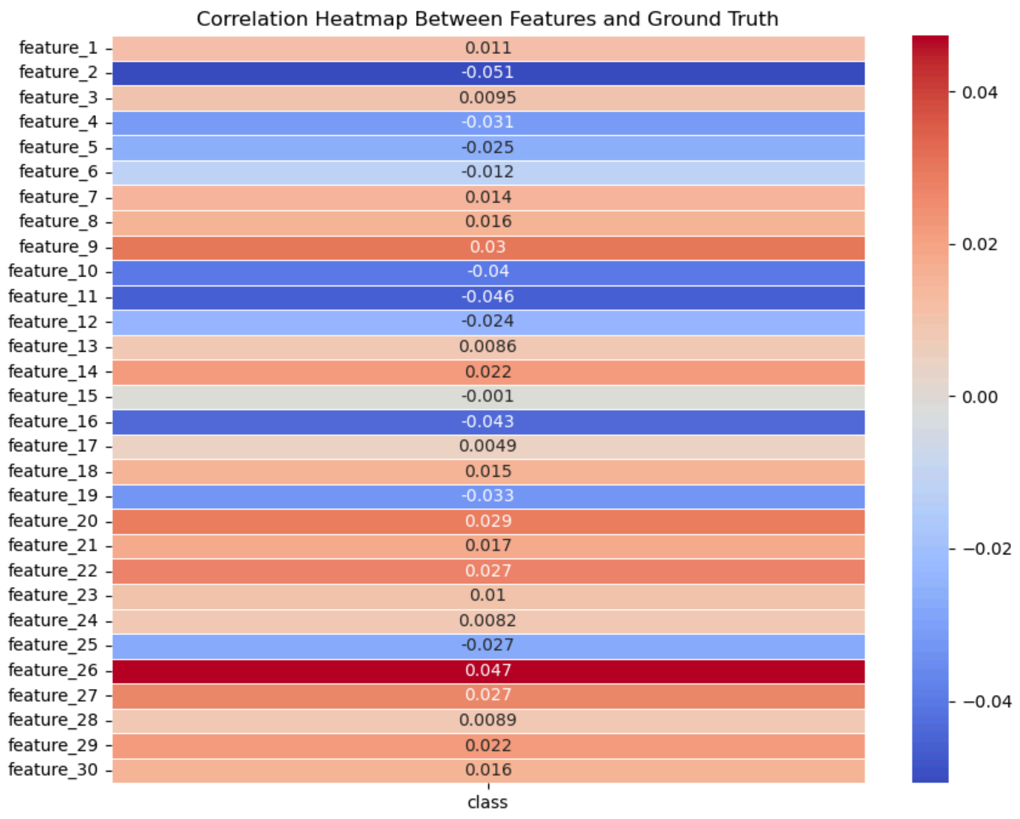


Figure 1: Correlation Heatmap between Words and Spam Label

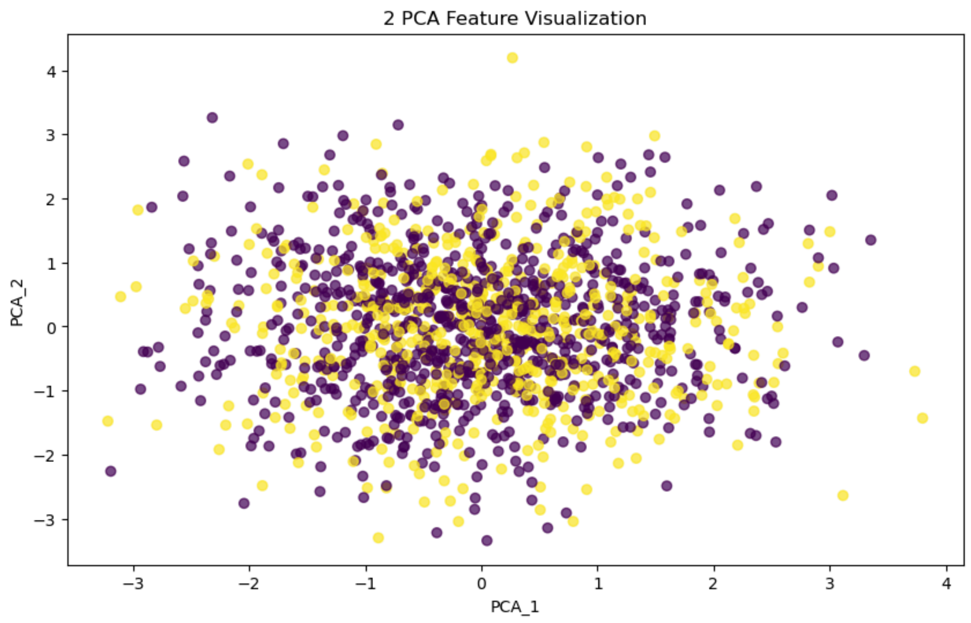
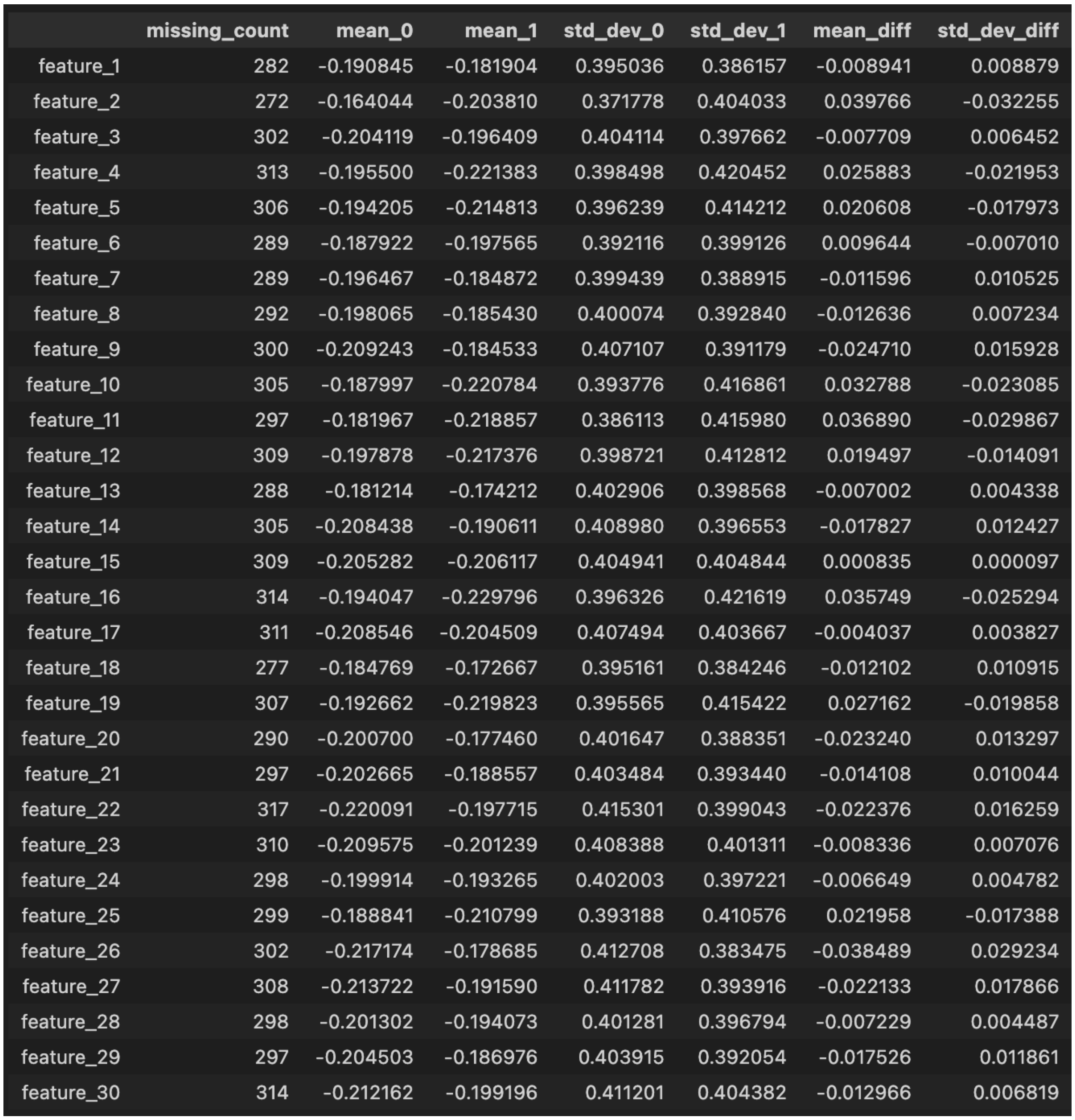


Figure 2: Exploratory Data Analysis PCA

Missing Values, Means and Standard Deviation by Class Label for Each Feature, and Difference



| | missing_count | mean_0 | mean_1 | std_dev_0 | std_dev_1 | mean_diff | std_dev_diff |
|---|---|---|---|---|---|---|---|
| feature_1 | 282 | -0.190845 | -0.181904 | 0.395036 | 0.386157 | -0.008941 | 0.008879 |
| feature_2 | 272 | -0.164044 | -0.203810 | 0.371778 | 0.404033 | 0.039766 | -0.032255 |
| feature_3 | 302 | -0.204119 | -0.196409 | 0.404114 | 0.397662 | -0.007709 | 0.006452 |
| feature_4 | 313 | -0.195500 | -0.221383 | 0.398498 | 0.420452 | 0.025883 | -0.021953 |
| feature_5 | 306 | -0.194205 | -0.214813 | 0.396239 | 0.414212 | 0.020608 | -0.017973 |
| feature_6 | 289 | -0.187922 | -0.197565 | 0.392116 | 0.399126 | 0.009644 | -0.007010 |
| feature_7 | 289 | -0.196467 | -0.184872 | 0.399439 | 0.388915 | -0.011596 | 0.010525 |
| feature_8 | 292 | -0.198065 | -0.185430 | 0.400074 | 0.392840 | -0.012636 | 0.007234 |
| feature_9 | 300 | -0.209243 | -0.184533 | 0.407107 | 0.391179 | -0.024710 | 0.015928 |
| feature_10 | 305 | -0.187997 | -0.220784 | 0.393776 | 0.416861 | 0.032788 | -0.023085 |
| feature_11 | 297 | -0.181967 | -0.218857 | 0.386113 | 0.415980 | 0.036890 | -0.029867 |
| feature_12 | 309 | -0.197878 | -0.217376 | 0.398721 | 0.412812 | 0.019497 | -0.014091 |
| feature_13 | 288 | -0.181214 | -0.174212 | 0.402906 | 0.398568 | -0.007002 | 0.004338 |
| feature_14 | 305 | -0.208438 | -0.190611 | 0.408980 | 0.396553 | -0.017827 | 0.012427 |
| feature_15 | 309 | -0.205282 | -0.206117 | 0.404941 | 0.404844 | 0.000835 | 0.000097 |
| feature_16 | 314 | -0.194047 | -0.229796 | 0.396326 | 0.421619 | 0.035749 | -0.025294 |
| feature_17 | 311 | -0.208546 | -0.204509 | 0.407494 | 0.403667 | -0.004037 | 0.003827 |
| feature_18 | 277 | -0.184769 | -0.172667 | 0.395161 | 0.384246 | -0.012102 | 0.010915 |
| feature_19 | 307 | -0.192662 | -0.219823 | 0.395565 | 0.415422 | 0.027162 | -0.019858 |
| feature_20 | 290 | -0.200700 | -0.177460 | 0.401647 | 0.388351 | -0.023240 | 0.013297 |
| feature_21 | 297 | -0.202665 | -0.188557 | 0.403484 | 0.393440 | -0.014108 | 0.010044 |
| feature_22 | 317 | -0.220091 | -0.197715 | 0.415301 | 0.399043 | -0.022376 | 0.016259 |
| feature_23 | 310 | -0.209575 | -0.201239 | 0.408388 | 0.401311 | -0.008336 | 0.007076 |
| feature_24 | 298 | -0.199914 | -0.193265 | 0.402003 | 0.397221 | -0.006649 | 0.004782 |
| feature_25 | 299 | -0.188841 | -0.210799 | 0.393188 | 0.410576 | 0.021958 | -0.017388 |
| feature_26 | 302 | -0.217174 | -0.178685 | 0.412708 | 0.383475 | -0.038489 | 0.029234 |
| feature_27 | 308 | -0.213722 | -0.191590 | 0.411782 | 0.393916 | -0.022133 | 0.017866 |
| feature_28 | 298 | -0.201302 | -0.194073 | 0.401281 | 0.396794 | -0.007229 | 0.004487 |
| feature_29 | 297 | -0.204503 | -0.186976 | 0.403915 | 0.392054 | -0.017526 | 0.011861 |
| feature_30 | 314 | -0.212162 | -0.199196 | 0.411201 | 0.404382 | -0.012966 | 0.006819 |

Figure 3: Mean and Standard Deviation of each Word Separated by Spam Label

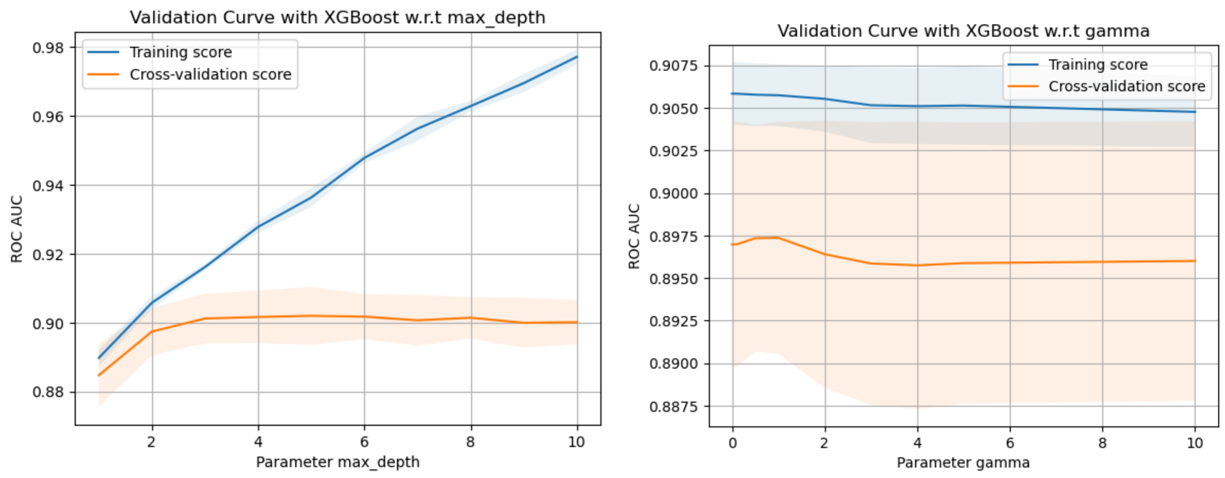


Figure 4: Validation Curves for Hyperparameter Tuning

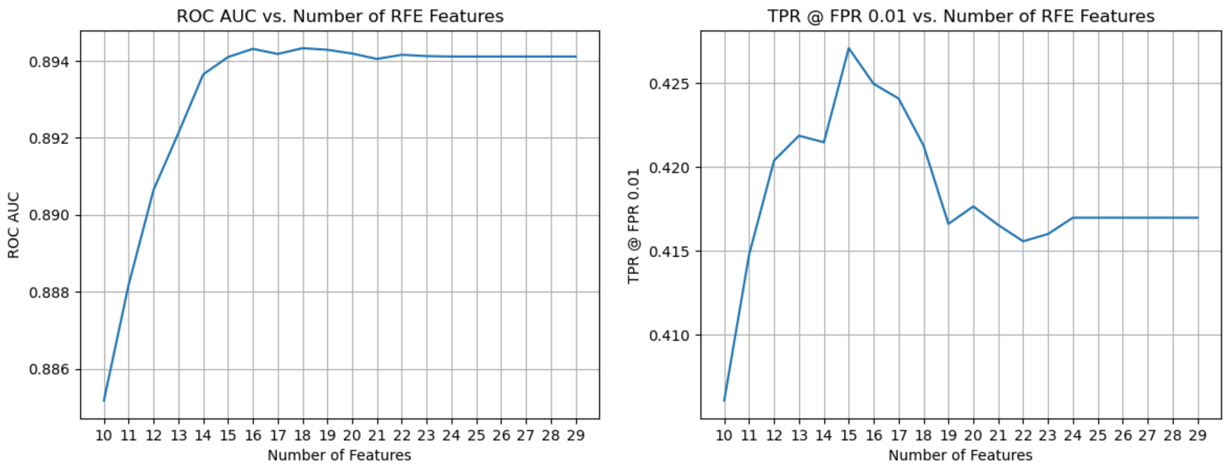


Figure 5: Average RFE Curves after Hyperparameter Tuning over 50 Train/Test Splits