The Binary Classification of Spam

GitHub URL
https://github.com/Crowmium1/UCDPA-L.J._Fitzgerald.git

Abstract
This report delves into the Spambase dataset to classify emails as spam or not. A rigorous feature engineering process refined the dataset, with descriptive statistics guiding transformations and feature selections. Four models were used and hyperparameter-tuned for optimal performance. Inferential statistics and hypothesis testing provided deeper insights into the relative efficiencies of these models. Furthermore, ensemble techniques were explored, culminating in a stacking classifier that aimed to harness the strengths of all individual models.

Introduction
This paper presents a comprehensive evaluation of various machine learning models in their ability to classify emails as 'spam' or 'not spam'. The study evaluates multiple models and aims to determine which model provides the most reliable and accurate email classification using accuracy as the chosen performance metric. Other diagnostic metrics, such as precision, F1 scores, and confusion matrices are addressed.

Email classification is a pivotal task to filter out unwanted messages and prioritize genuine communication. With the continuous growth of electronic mail, automated systems need to be adept at correctly classifying 'spam' and 'not spam' emails.

The machine learning models being evaluated are as follows:

- **K-Nearest Neighbors (KNN)**: A non-parametric method that classifies based on the majority label of its nearest data points.
- **Support Vector Machine (SVM)**: A linear classifier that determines an optimal hyperplane to segregate classes.
- **Random Forest**: An ensemble of decision trees that considers random subsets of features.
- **Gradient Boosting (GB)**: An ensemble method that builds trees sequentially to correct the errors of its predecessor.
- **Bernoulli Naive Bayes (BNB)**: A probabilistic classifier tailored for Binary/Boolean features.
- **Gaussian Naive Bayes (GNB)**: A probabilistic classifier that assumes features follow a normal distribution.
- **Ada Boost (AB)**: An ensemble method that adjusts weights iteratively.
- **Stacked Classifier**: An ensemble approach combining multiple models.

Dataset
For our study, we sourced a dataset containing labelled emails. Non-spam email origins are from a person's work and personal sources. There are 58 feature-labelled columns with 4601 rows. 1 column is the 'spam/ham' target variable, where 1 is spam and 0 is not spam. The other 57 columns consist of word and character count frequency information. This information is taken from the

‘Spambase’ dataset, found on the UCI repository (see references). There are missing values in this dataset.

Implementation Process

Data Import and Preparation:

1. Modules are imported and functions are kept at the start of the project.
2. Data is read and combined from the datasets into numpy arrays. Columns are checked for consistency.
3. Conducted descriptive statistics and visualized the spam/ham class distribution.
4. Data structures, duplicates, and missing values were managed.
5. Explored the distribution of duplicated rows relative to the target variable.
6. Visualized data correlation through heatmaps and identified high-correlation features.

Feature Engineering:

1. Employed principal component analysis (PCA) on the dataset for dimensionality reduction.
2. Developed a dataframe for correlated feature labels. Utilized Random Forest and Gradient Boosting classifiers to derive feature importance values, followed by conversions to dataframe format.
3. Visualized performance metrics of the model against the number of features used, along with ROC curves and reduced dimensionality accordingly.
4. Conducted descriptive statistics on each feature-engineered dataframe.

Model Evaluation and Hyper-tuning:

1. Evaluated base model performance using default parameters, ensuring SVM and KNN models were scaled appropriately.
2. Split the data into training and test subsets, applying cross-validation.
3. Implemented loops to train, fit, and predict data, documenting accuracy scores.
4. Generated ROC curves on base data and calculated confidence intervals for accuracy.
5. Iterated the process with hyper-tuned parameters and stored the best models for ensemble training and stacking.
6. Evaluated performance of KNN base and hyper-tuned models using hypothesis testing.

Ensemble Methods:

1. Re-split the data and evaluated models on validation data, storing accuracies for hypothesis testing.
2. Printed confusion matrices and classification reports for each model trained on optimal parameters.
3. Visualized model performances and reported validation set scores.

4. Conducted bootstrap confidence intervals on data and formed three null hypotheses concerning model performances.
5. Undertook paired T-tests to determine significant performance differences between models.

Stacking:

1. Formed a null hypothesis regarding the stacked model's performance relative to individual ensemble models.
2. Used ensemble models as estimators and instantiated a stacking classifier.
3. Trained and evaluated data on the validation set.
4. Performed a paired T-test comparing the stacked model with the best-performing ensemble method.

Analysis Plots:

1. Diagnosed model performance, identifying underfitting, overfitting, or potential data needs with learning curve plots.
2. Employed calibration curves to inspect prediction probability alignment.

Image Extraction and File Handling:

1. Archived the trained model for future access.
2. Transformed the script into an HTML format for improved accessibility.
3. Extracted embedded script images for independent storage.

Results

The descriptive statistics: The original Dataframe gave an indication for the size, shape, and frequency of the data, and the distribution of the target class (fig. 1-3). Duplicates' distribution seemed to match the true target variable distribution (fig. 4); therefore, I believe these emails were not explicitly spam.

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	\
count	4601.000000	4601.000000	4601.000000	4601.000000	
mean	0.104553	0.213015	0.280656	0.065425	
std	0.305358	1.290575	0.504143	1.395151	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	0.000000	0.000000	0.420000	0.000000	
max	4.540000	14.280000	5.100000	42.810000	

	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	\
count	4601.000000	4601.000000	4601.000000	4601.000000	
mean	0.312223	0.095901	0.114208	0.105295	
std	0.672513	0.273824	0.391441	0.401071	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.000000	0.000000	0.000000	0.000000	
50%	0.000000	0.000000	0.000000	0.000000	
75%	0.380000	0.000000	0.000000	0.000000	
max	10.000000	5.880000	7.270000	11.110000	

	word_freq_order	word_freq_mail	...	char_freq_;	char_freq_(	\
count	4601.000000	4601.000000	...	4601.000000	4601.000000	
mean	0.090067	0.239413	...	0.038575	0.139030	
std	0.278616	0.644755	...	0.243471	0.270355	
min	0.000000	0.000000	...	0.000000	0.000000	
...						
75%	266.000000	1.000000				
max	15841.000000	1.000000				

Fig.1: Spambase descriptive statistics

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	...	capital_run_length_average	capital_run_length_longest	capital_run_length_total	spam/ham
0	0.00	0.64	0.64	0.0	...	3.756	61	278	1
1	0.21	0.28	0.50	0.0	...	5.114	101	1028	1
2	0.06	0.00	0.71	0.0	...	9.821	485	2259	1
3	0.00	0.00	0.00	0.0	...	3.537	40	191	1
4	0.00	0.00	0.00	0.0	...	3.537	40	191	1

Fig.2: Spambase Dataframe

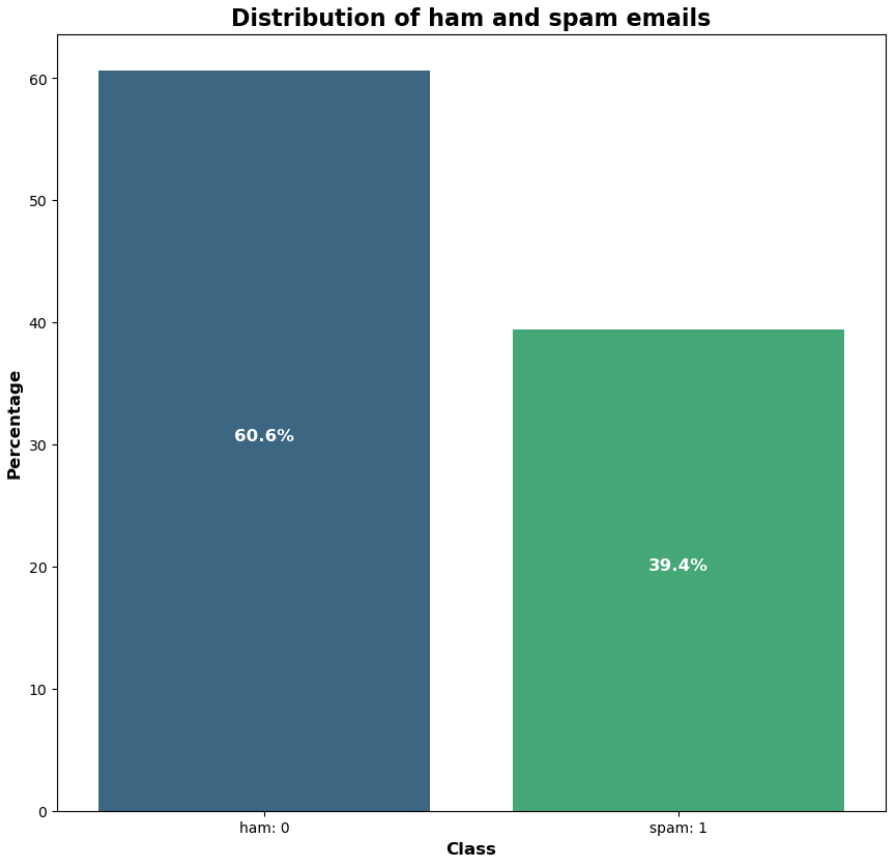


Fig. 3: Distribution of target class variable in original dataframe.

```
Distribution for duplicate rows with max value <= 3:
0: 21 (100.00%)


Distribution for duplicate rows with max value <= 100:
0: 238 (86.55%)
1: 37 (13.45%)


Distribution for duplicate rows with max value of 15841:
0: 257 (65.73%)
1: 134 (34.27%)


Overall distribution:
0: 2788 (60.60%)
1: 1813 (39.40%)
```

Fig. 4: Distribution of duplicate values vs. target class

For dimensionality reduction, a correlation matrix (fig.5) and principal component analysis (fig.6) were applied to the Spambase dataset. Highly correlated features with the target variable were deemed valuable. 49 of the original dataset's feature labels comprised 95% of the explained variance.

Out of the three engineered feature variables, the random forest classified feature importance's showed the best performance on ROC AUCs for the validation set and had the highest TPR and lowest FPR (fig.7). Comparisons between random forest and gradient boosting importance values revealed similarities, with random forest having a lower standard deviation (fig.8-10). Consequently, 30 features were selected for the random forest feature variable for further analysis.

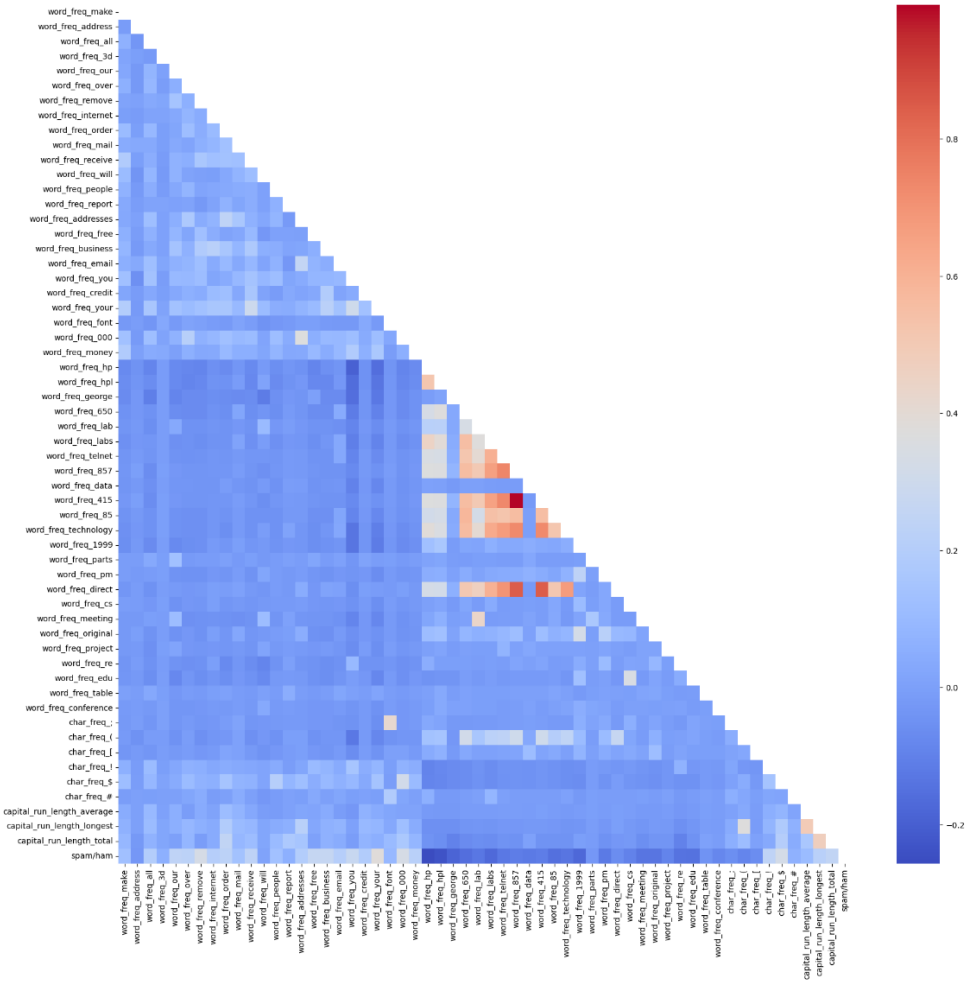


Fig. 5: Correlated feature labels

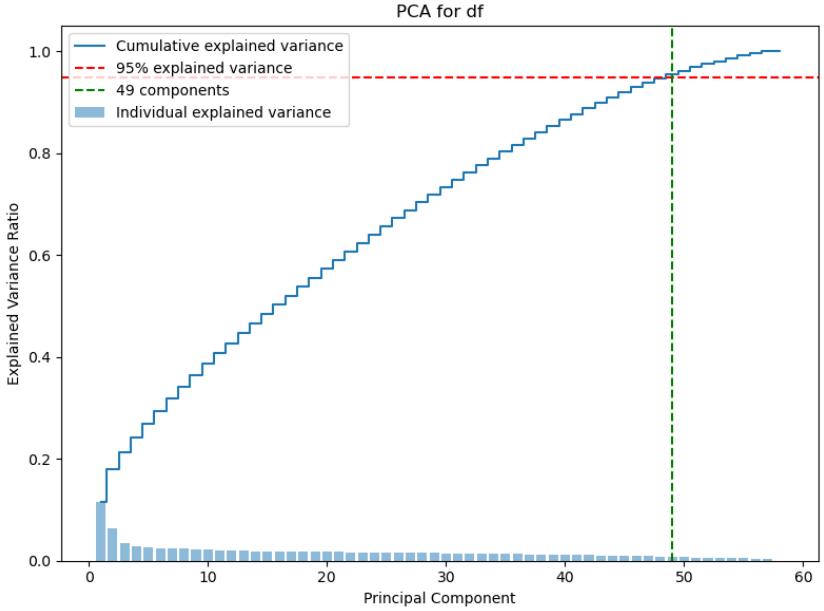
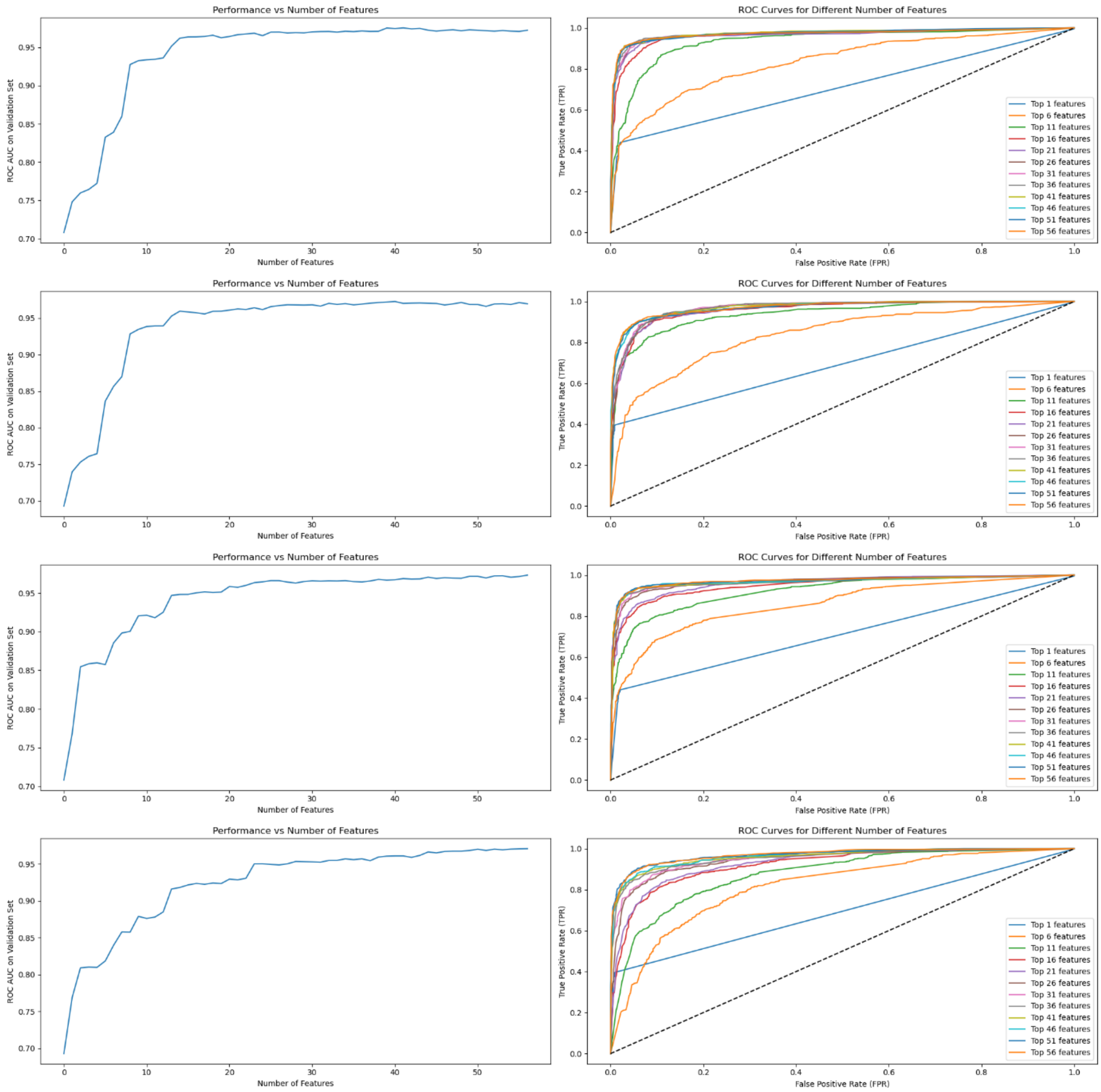


Fig. 6: PCA on original dataset


Performance vs Number of Features
ROC AUC on Validation Set
Number of Features
ROC Curves for Different Number of Features
True Positive Rate (TPR)
False Positive Rate (FPR)
Top 1 features
Top 6 features
Top 11 features
Top 16 features
Top 21 features
Top 26 features
Top 31 features
Top 36 features
Top 41 features
Top 46 features
Top 51 features
Top 56 features

```
Processing for: Count Features
```

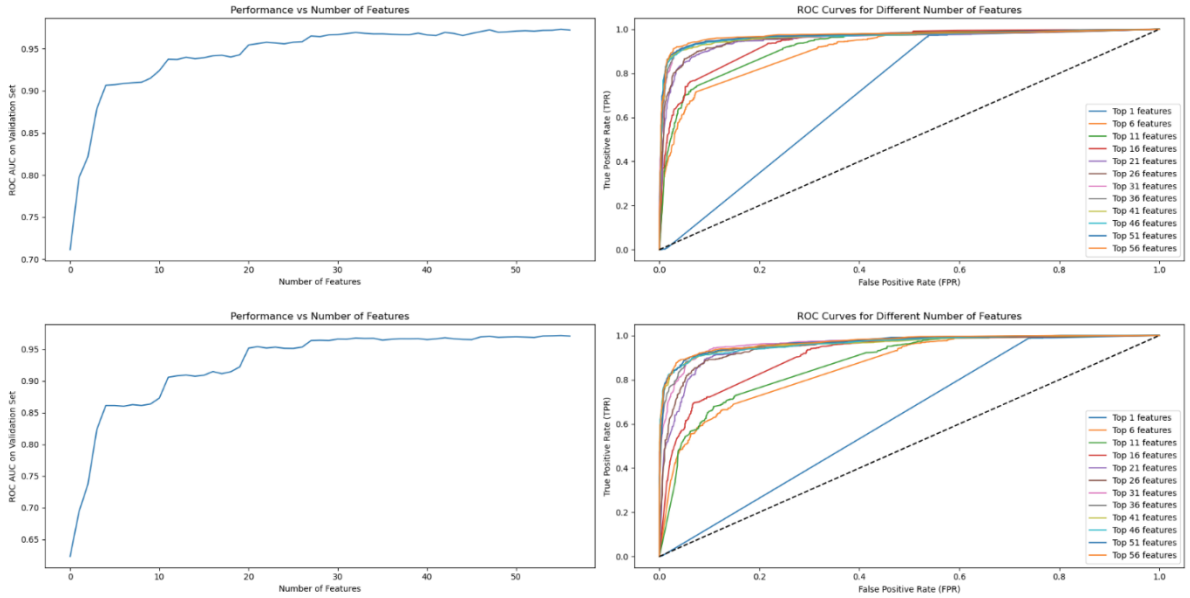


Fig. 7: Performance vs. number of features and ROC curves for feature variables.

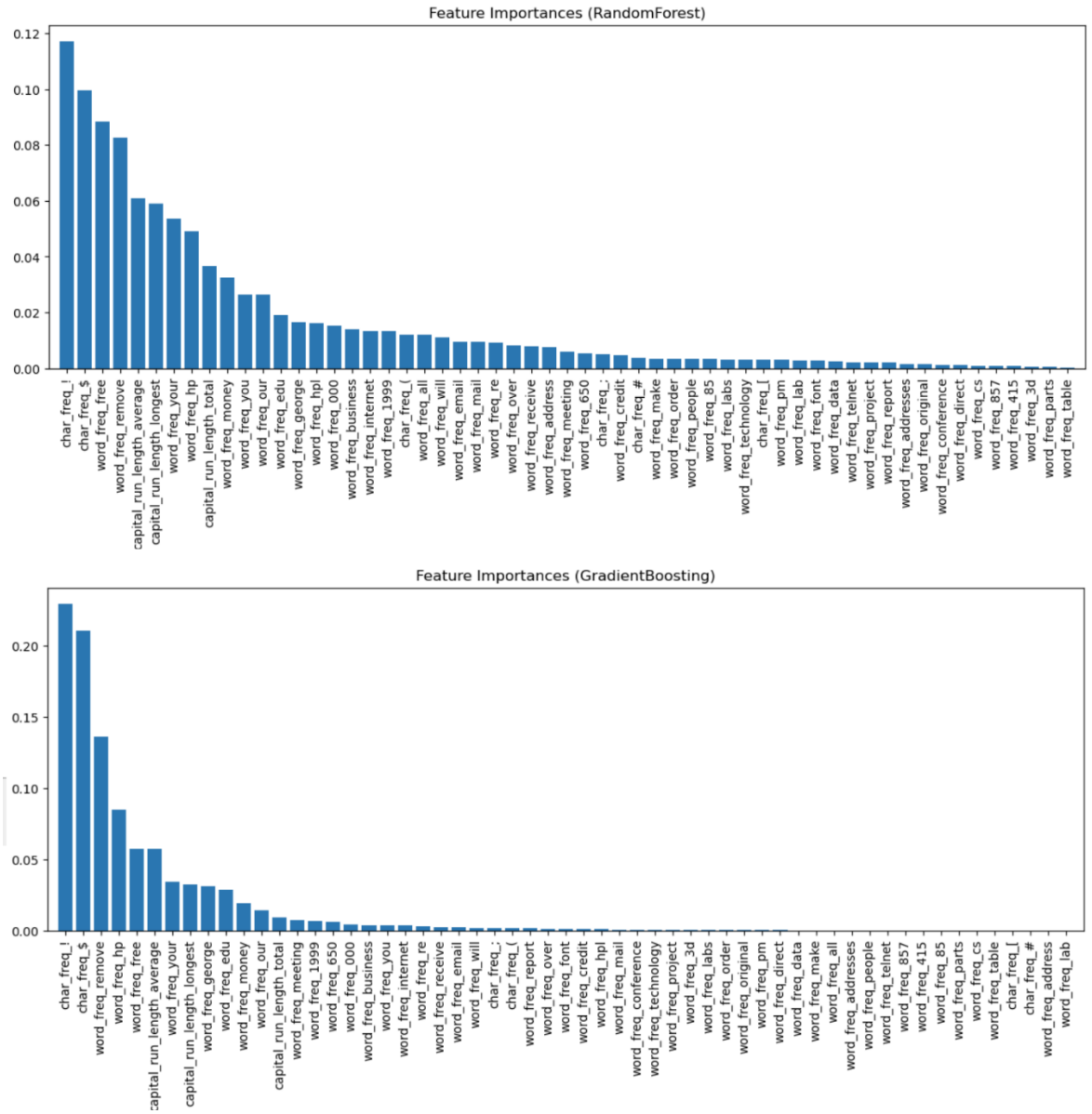


Fig.8: Feature Importances

	feature_label	RF_Importance
0	char_freq_!	0.119398
1	char_freq_$	0.090807
2	word_freq_remove	0.077554
3	word_freq_free	0.068374
4	capital_run_length_average	0.067342
5	capital_run_length_longest	0.063396
6	word_freq_your	0.059592
7	word_freq_hp	0.043141
8	capital_run_length_total	0.040351
9	word_freq_money	0.034100
10	word_freq_you	0.030029
11	word_freq_our	0.028693
12	word_freq_george	0.021890
13	word_freq_edu	0.019250
14	word_freq_hpl	0.019107
15	word_freq_000	0.018947
16	word_freq_business	0.013037
17	word_freq_internet	0.012980
18	word_freq_receive	0.012465
19	char_freq_(	0.012235
20	word_freq_1999	0.011712
21	word_freq_will	0.011171
22	word_freq_re	0.009451
23	word_freq_email	0.008057
24	word_freq_credit	0.008012
25	word_freq_all	0.007704
26	word_freq_mail	0.007657
27	word_freq_over	0.007484
28	word_freq_meeting	0.005817
29	char_freq_;	0.005651

	feature_label	GB_Importance
0	char_freq_!	0.229453
1	char_freq_$	0.210535
2	word_freq_remove	0.136491
3	word_freq_hp	0.084696
4	word_freq_free	0.057394
5	capital_run_length_average	0.056901
6	word_freq_your	0.034335
7	capital_run_length_longest	0.032424
8	word_freq_george	0.030896
9	word_freq_edu	0.028372
10	word_freq_money	0.019010
11	word_freq_our	0.014104
12	capital_run_length_total	0.009286
13	word_freq_meeting	0.007308
14	word_freq_1999	0.006462
15	word_freq_650	0.005872
16	word_freq_000	0.004455
17	word_freq_you	0.003576
18	word_freq_business	0.003549
19	word_freq_internet	0.003364
20	word_freq_re	0.003092
21	word_freq_receive	0.002507
22	word_freq_email	0.002193
23	word_freq_will	0.001719
24	char_freq_;	0.001693
25	char_freq_(	0.001488
26	word_freq_report	0.001487
27	word_freq_over	0.001278
28	word_freq_font	0.001110
29	word_freq_credit	0.001018

	feature_label	High Correlation Count
27	word_freq_650	11
31	word_freq_857	11
33	word_freq_415	11
39	word_freq_direct	10
35	word_freq_technology	10
29	word_freq_labs	10
30	word_freq_telnet	10
34	word_freq_85	10
25	word_freq_hpl	9
28	word_freq_lab	9
24	word_freq_hp	9
49	char_freq_(	4
55	capital_run_length_longest	3
22	word_freq_000	2
40	word_freq_cs	1
36	word_freq_1999	1
41	word_freq_meeting	1
42	word_freq_original	1
45	word_freq_edu	1
48	char_freq_;	1
52	char_freq_$	1
54	capital_run_length_average	1
56	capital_run_length_total	1
14	word_freq_addresses	1
20	word_freq_your	1
21	word_freq_font	1
18	word_freq_you	1
23	word_freq_money	0
44	word_freq_re	0
7	word_freq_internet	0

Fig. 9: Feature Variables

```
RF Features Shape of DataFrame with selected features: (30, 2)

RF Features Descriptive Statistics of the DataFrame with selected features:

       RF_Importance
count      30.000000
mean        0.031180
std         0.029621
min         0.005651
25%         0.009881
50%         0.019027
75%         0.042443
max         0.119398


--------------------------------------------------
GB Features Shape of DataFrame with selected features: (30, 2)

GB Features Descriptive Statistics of the DataFrame with selected features:

       GB_Importance
count      30.000000
mean        0.033202
std         0.059042
min         0.001018
25%         0.002271
50%         0.006167
75%         0.032042
max         0.229453


--------------------------------------------------
Count Features Shape of DataFrame with selected features: (30, 2)

Count Features Descriptive Statistics of the DataFrame with selected features:

       High Correlation Count
count               30.000000
mean                 4.400000
std                  4.422513
min                  0.000000
25%                  1.000000
50%                  1.000000
75%                  9.750000
max                 11.000000


--------------------------------------------------
```

Fig.10: Descriptive Statistics for feature variables.

The inferential statistics: Four base models were evaluated, and the SVM model outperformed in terms of accuracy, precision, recall, and F1 score. The confidence intervals indicate results falling in that range 95% of the time (fig.12).

After hyper-tuning, SVM and KNN showed slight accuracy improvements (fig.13). Hyper-Tuned SVM exhibited high specificity without compromising sensitivity. GNB and BNB saw no changes after hyper tuning, due to limited tuning of hyperparameters. All models were likely quite well-tuned already, as the baseline performance was already very high.

The ROC AUC curves (fig.14) between base and hyper-tuned models show negligible differences.

False Positive (FP) *or type 1 error or sensitivity: A genuine (not spam) email is classified as spam. This means an important email might get missed or deleted.*

False Negative (FN) *or type 2 error or specificity: A spam email is classified as genuine. This means the user gets unwanted emails in their inbox.*

The cost of an FP might be higher if a critical email is missed, especially in a business context, while the cost of an FN might be lower since it would mean a minor inconvenience of deleting or marking the email as spam manually. Therefore, in terms of balancing sensitivity and specificity, we can aim for a higher specificity (true negative rate). The cost of missing a genuine email could be higher than the inconvenience of manually sorting out a spam email that lands in the inbox.

	Model	TPR	FPR	TNR	FNR
0	Base KNN	0.862259	0.046595	0.953405	0.137741
1	Base SVM	0.873278	0.035842	0.964158	0.126722
2	Base GNB	0.884298	0.059140	0.940860	0.115702
3	Base BNB	0.790634	0.064516	0.935484	0.209366
4	Hyper-Tuned KNN	0.867769	0.044803	0.955197	0.132231
5	Hyper-Tuned SVM	0.881543	0.034050	0.965950	0.118457
6	Hyper-Tuned GNB	0.884298	0.059140	0.940860	0.115702
7	Hyper-Tuned BNB	0.790634	0.064516	0.935484	0.209366

Fig.11: TPR, FPR, TNR, FNR

Notably, both optimized KNN and SVM models have slightly improved in correctly identifying 'spam' emails with a higher TPR and not misclassifying 'not spam' emails as 'spam' with a lower FPR (fig.11 and 15). The optimized SVM model performs the best in terms of highest TPR and TNR and lowest FPR and FNR.

Hypothesis 1: The obtained p-value comparing hyper-tuned KNN to the base KNN was 0.6124, failing to show a significant improvement, where alpha=0.025. In this case, we failed to reject the null hypothesis and there is no statistically significant improvement in performance from the base model to the hyper-tuned model. The t-statistic of -0.5069 and effect size of 0.0119 corroborate this (fig.16).

```
Evaluating KNN...

Accuracy: 0.9174809989142236

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.95      0.93       558
           1       0.92      0.86      0.89       363

    accuracy                           0.92       921
   macro avg       0.92      0.91      0.91       921
weighted avg       0.92      0.92      0.92       921


Confusion Matrix:
[[532  26]
 [ 50 313]]
```

```
Evaluating GNB...

Accuracy: 0.9185667752442996

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       558
           1       0.91      0.88      0.90       363

    accuracy                           0.92       921
   macro avg       0.92      0.91      0.91       921
weighted avg       0.92      0.92      0.92       921


Confusion Matrix:
[[525  33]
 [ 42 321]]
```

```
Evaluating SVM...

Accuracy: 0.9283387622149837

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       558
           1       0.94      0.87      0.91       363

    accuracy                           0.93       921
   macro avg       0.93      0.92      0.92       921
weighted avg       0.93      0.93      0.93       921


Confusion Matrix:
[[538  20]
 [ 46 317]]
```

```
Evaluating BNB...

Accuracy: 0.8783930510314875

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.90       558
           1       0.89      0.79      0.84       363

    accuracy                           0.88       921
   macro avg       0.88      0.86      0.87       921
weighted avg       0.88      0.88      0.88       921


Confusion Matrix:
[[522  36]
 [ 76 287]]
```

```
X Train shape: (3680, 30)
X Val shape: (921, 30)
y Train shape: (3680,)
y Val shape: (921,)
```

```
Computing confidence interval for KNN...
95% Confidence Interval for KNN Accuracy: [0.900, 0.935]
Computing confidence interval for SVM...
95% Confidence Interval for SVM Accuracy: [0.911, 0.944]
Computing confidence interval for GNB...
95% Confidence Interval for GNB Accuracy: [0.900, 0.935]
Computing confidence interval for BNB...
95% Confidence Interval for BNB Accuracy: [0.857, 0.899]
```

Fig. 12: Inferential Statistics for Base Models

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

Evaluating KNN with optimized parameters...

Accuracy: 0.9207383279044516

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       558
           1       0.93      0.87      0.90       363

    accuracy                           0.92       921
   macro avg       0.92      0.91      0.92       921
weighted avg       0.92      0.92      0.92       921


Confusion Matrix:
[[533  25]
 [ 48 315]]
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

Evaluating GNB with optimized parameters...

Accuracy: 0.9185667752442996

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.94      0.93       558
           1       0.91      0.88      0.90       363

    accuracy                           0.92       921
   macro avg       0.92      0.91      0.91       921
weighted avg       0.92      0.92      0.92       921


Confusion Matrix:
[[525  33]
 [ 42 321]]
```

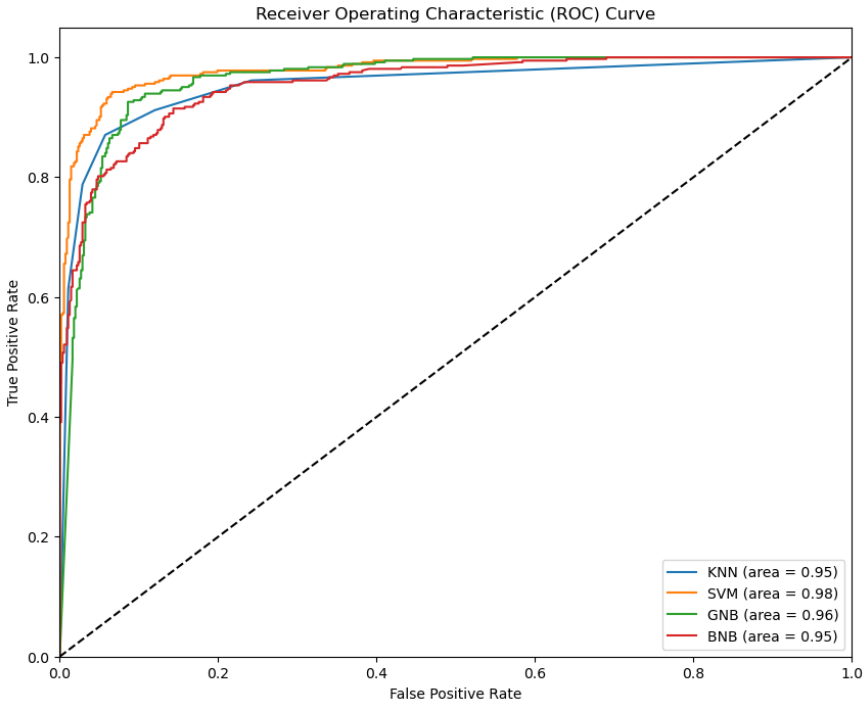
```
Fitting 5 folds for each of 3 candidates, totalling 15 fits

Evaluating SVM with optimized parameters...

Accuracy: 0.9326818675352877

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       558
           1       0.94      0.88      0.91       363

    accuracy                           0.93       921
   macro avg       0.94      0.92      0.93       921
weighted avg       0.93      0.93      0.93       921


Confusion Matrix:
[[539  19]
 [ 43 320]]
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits

Evaluating BNB with optimized parameters...

Accuracy: 0.8783930510314875

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.94      0.90       558
           1       0.89      0.79      0.84       363

    accuracy                           0.88       921
   macro avg       0.88      0.86      0.87       921
weighted avg       0.88      0.88      0.88       921


Confusion Matrix:
[[522  36]
 [ 76 287]]
```

```
X Train shape: (3680, 30)
X Val shape: (921, 30)
y Train shape: (3680,)
y Val shape: (921,)
```

```
Computing confidence interval for KNN...
95% Confidence Interval for KNN Accuracy: [0.904, 0.938]
Computing confidence interval for SVM...
95% Confidence Interval for SVM Accuracy: [0.916, 0.949]
Computing confidence interval for GNB...
95% Confidence Interval for GNB Accuracy: [0.899, 0.936]
Computing confidence interval for BNB...
95% Confidence Interval for BNB Accuracy: [0.858, 0.899]
```

Fig. 13: Inferential Statistics for hyper-tuned Models



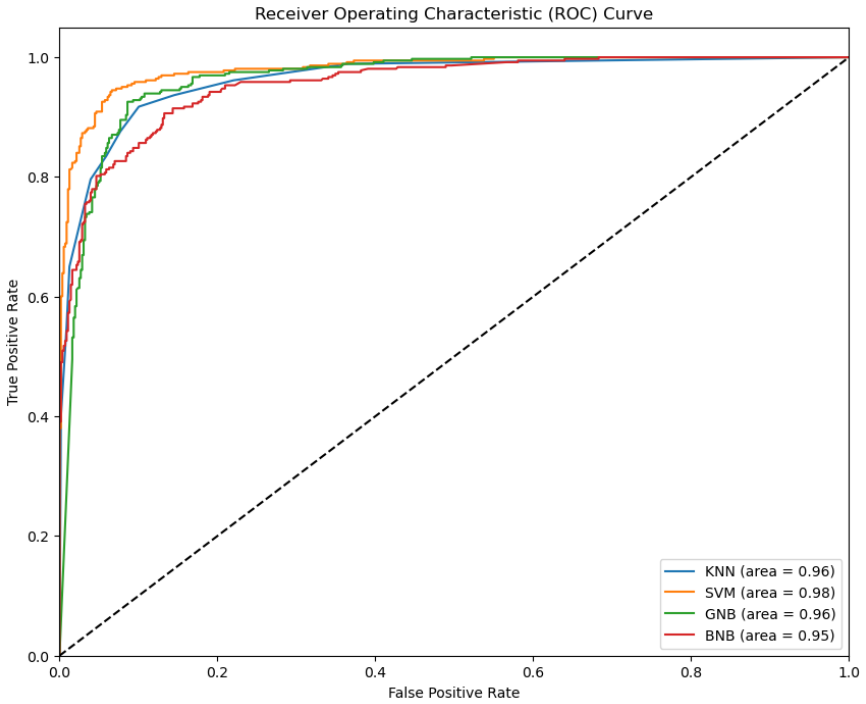
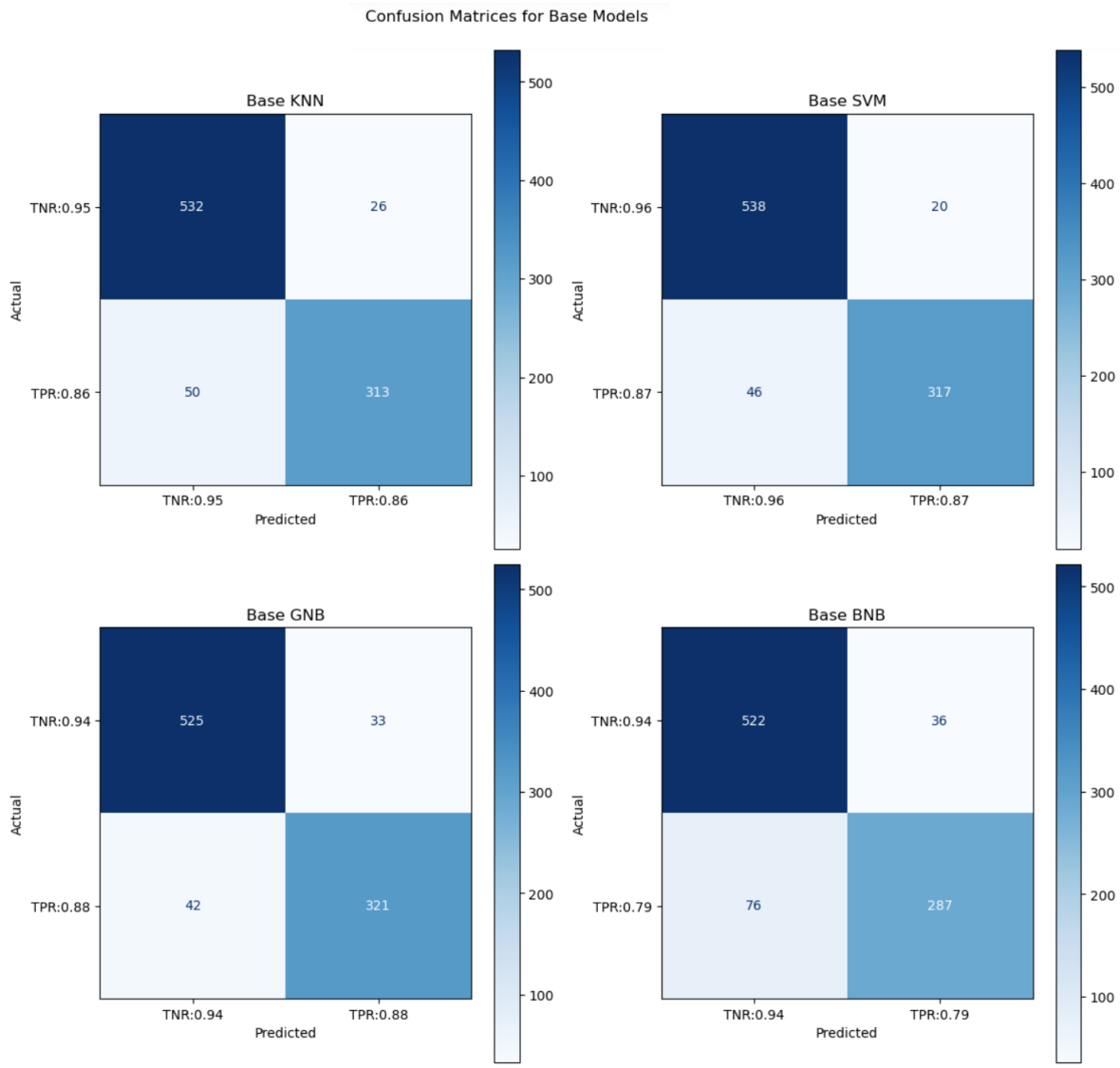


Fig. 14: ROC curves for base vs. hyper-tuned models


Confusion Matrices for Base Models
Base KNN
TNR:0.95
TPR:0.86
532
26
50
313
Actual
Predicted
TNR:0.95
TPR:0.86
Base SVM
TNR:0.96
TPR:0.87
538
20
46
317
Actual
Predicted
TNR:0.96
TPR:0.87
Base GNB
TNR:0.94
TPR:0.88
525
33
42
321
Actual
Predicted
TNR:0.94
TPR:0.88
Base BNB
TNR:0.94
TPR:0.79
522
36
76
287
Actual
Predicted
TNR:0.94
TPR:0.79
500
400
300
200
100

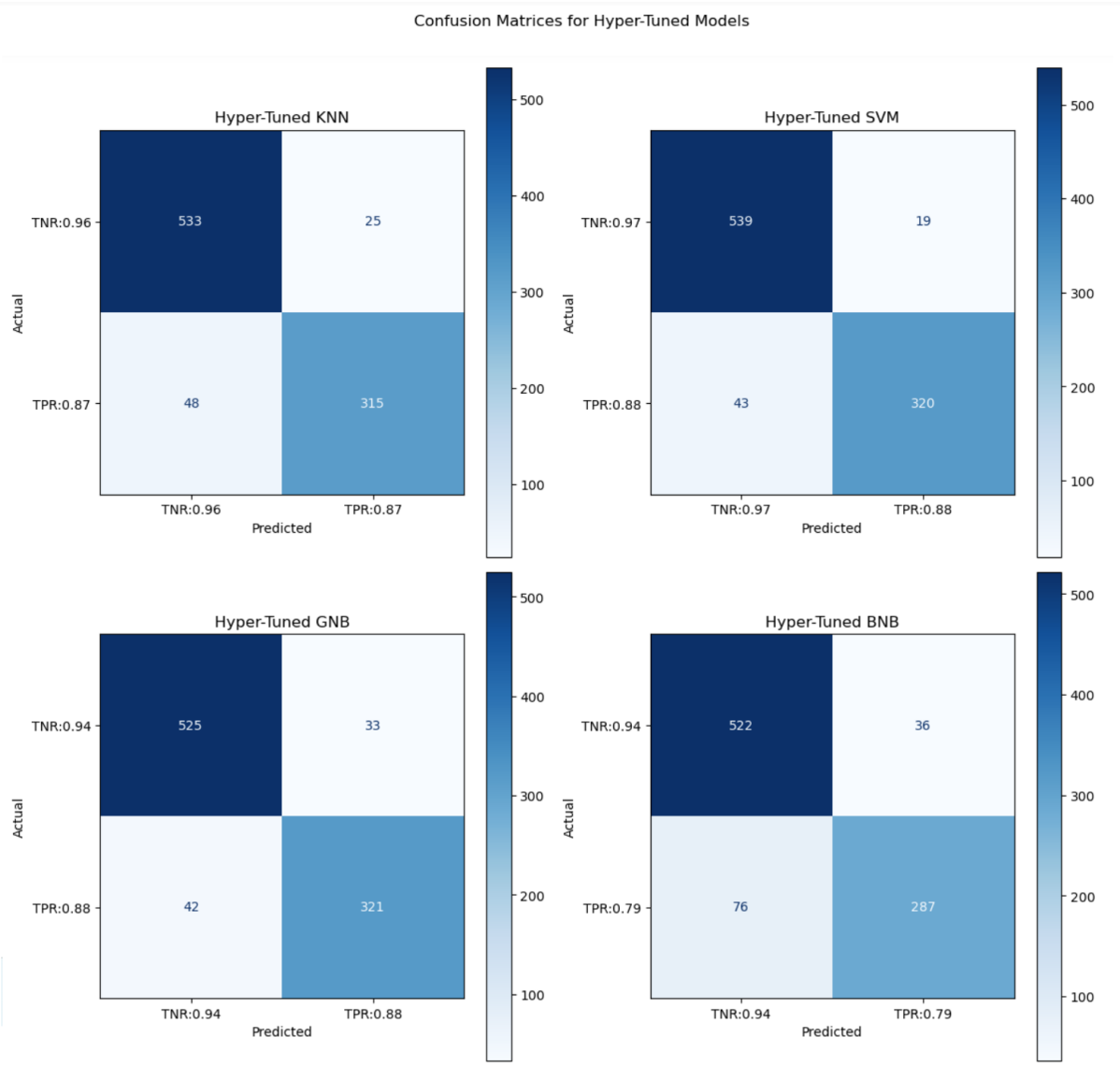


Fig. 15: Confusion matrices

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier', KNeighborsClassifier())])
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier', KNeighborsClassifier(n_neighbors=11))])
KNN

Objective: To determine if hypertuned KNN offers superior performance to
the base KNN in terms of accuracy.

H0: The performance of the hypertuned KNN is equivalent to the base KNN.

H1: The performance of the hypertuned KNN is significantly better than the base KNN.

p-value: 0.61235490600884

KNN (Base) vs KNN (Hypertuned):
T-statistic: -0.5069
P-value: 0.6124
Effect size: 0.0119

There's no significant performance difference between the hypertuned and the base KNN at the 2.5% level.

Conclusion:
In this analysis, base and hypertuned classifiers for KNN were trained. Performances were evaluated and
statistically compared. The results provide insights into whether hypertuning offers additional value
compared to the base model settings.
```

Fig. 16: Hypothesis 1

All ensemble models had similar accuracy scores of about 93%. Confusion matrices revealed more false negatives than false positives for all models, with Ada boost underperforming in precision. The F1 scores indicated a balanced performance between precision and recall for all models (fig.17 and 18). Paired t-tests comparing model pairs showed no significant performance differences (fig.19).

Observing the confusion matrices, all models have more false negatives than false positives, meaning they are more likely to miss 'spam' emails than to misclassify 'not spam' emails as 'spam'.

The random forest classifier has more false negatives and fewer false positives than AB and GB, meaning it misses a higher proportion of actual spam. This is preferable to important emails going straight to junk (fig. 18).

All models demonstrate a high degree of confidence, with overlapping intervals, indicating comparable performance. This suggests that there isn't overfitting, as the data is generalizing well to new data (fig.18).

The p-values from paired t-tests reveal no significant difference in performance between any of the model pairs (AB vs RF, AB vs GB, or RF vs GB). The high p-values (>0.05) lead to failing to reject the null hypotheses. The effect sizes, which quantify the magnitude of the difference between models, are near zero, reiterating the lack of any substantial difference in performances (fig.19).

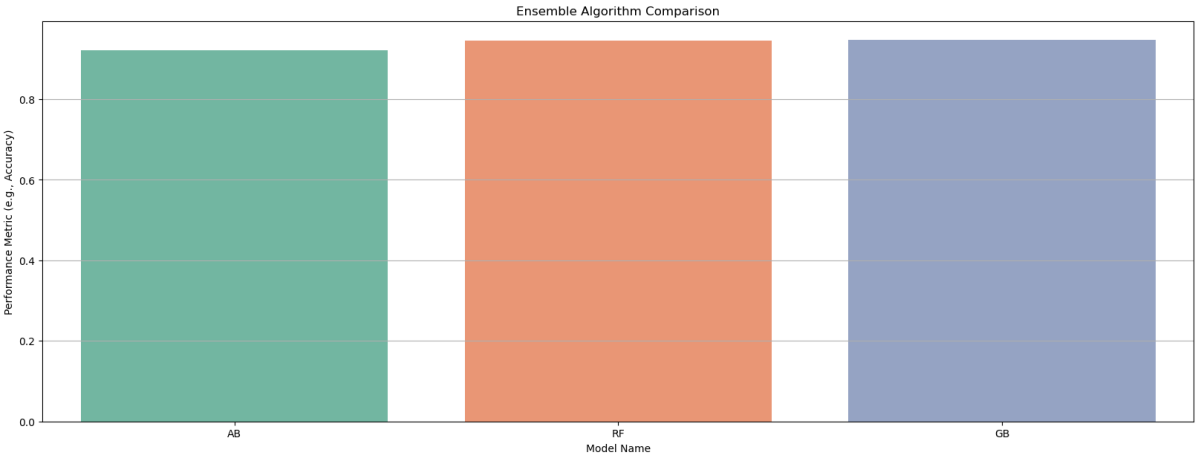


Fig. 17: Ensemble model performance

```
Accuracy: 0.9337676438653637

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       558
           1       0.94      0.89      0.91       363

    accuracy                           0.93       921
   macro avg       0.93      0.93      0.93       921
weighted avg       0.93      0.93      0.93       921


Confusion Matrix:
[[536  22]
 [ 39 324]]

Accuracy: 0.9348534201954397

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.97      0.95       558
           1       0.96      0.87      0.91       363

    accuracy                           0.93       921
   macro avg       0.94      0.92      0.93       921
weighted avg       0.94      0.93      0.93       921


Confusion Matrix:
[[544  14]
 [ 46 317]]

Accuracy: 0.9337676438653637

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.96      0.95       558
           1       0.94      0.89      0.91       363

    accuracy                           0.93       921
   macro avg       0.93      0.93      0.93       921
weighted avg       0.93      0.93      0.93       921


Confusion Matrix:
[[537  21]
 [ 40 323]]
```

```
X Train shape: (3680, 30)
X Test shape: (921, 30)
y Train shape: (3680,)
y Test shape: (921,)

Performance on Validation Set:
AB: 0.9338 Accuracy
RF: 0.9349 Accuracy
GB: 0.9338 Accuracy

Cross-Validation Scores:
AB: 0.9367 (+/- 0.0074)
RF: 0.9402 (+/- 0.0102)
GB: 0.9429 (+/- 0.0065)

Bootstrap Confidence Intervals:
AB: 91.75% - 95.01%
RF: 91.85% - 95.01%
GB: 91.64% - 94.79%
```

Fig.18: Ensemble Model Inferential Statistics

```
Objective: To determine if AB's performance is significantly different than RF in terms of accuracy.

H0: AB's performance is equivalent to RF.
H1: AB's performance is significantly different than RF.

--------------------------------------------------
Objective: To determine if AB's performance is significantly different than GB in terms of accuracy.

H0: AB's performance is equivalent to GB.
H1: AB's performance is significantly different than GB.

--------------------------------------------------
Objective: To determine if RF's performance is significantly different than GB in terms of accuracy.

H0: RF's performance is equivalent to GB.
H1: RF's performance is significantly different than GB.

--------------------------------------------------

Paired t-tests and Effect Sizes:
AB vs RF:
p-value: 0.8887273963242758
Effect Size: 0.004380382312778977
------------------------------
AB vs GB:
p-value: 1.0
Effect Size: 0.0
------------------------------
RF vs GB:
p-value: 0.8475109348450016
Effect Size: -0.004380382312778977
------------------------------

Conclusion:
In this analysis, ensemble classifiers were trained and their performance was evaluated using validation accuracy,
 cross-validation scores, bootstrapped confidence intervals, paired t-tests, and effect sizes. The results provide insights
into the relative performance of the models and their statistical differences.
```

Fig.19: Hypothesis 2

Learning curves are plotted for the entire dataset, while calibration and ROC curves target the validation set. These curves diagnose model bias and variance, providing insights into potential overfitting or underfitting.

The random forest classifier is showing slight overfitting as the test curves fail to fully converge with the training curves and there is a fair bit of variation between them after they plateau. This may be due to model complexity and suggests the model doesn't generalize well to unseen data. Simply adding more data is unlikely to enhance the model's performance. Instead, increasing the model's complexity by introducing new richer features may offer improvement. Ada boost and gradient boosting classifiers are well-balanced (fig. 20).

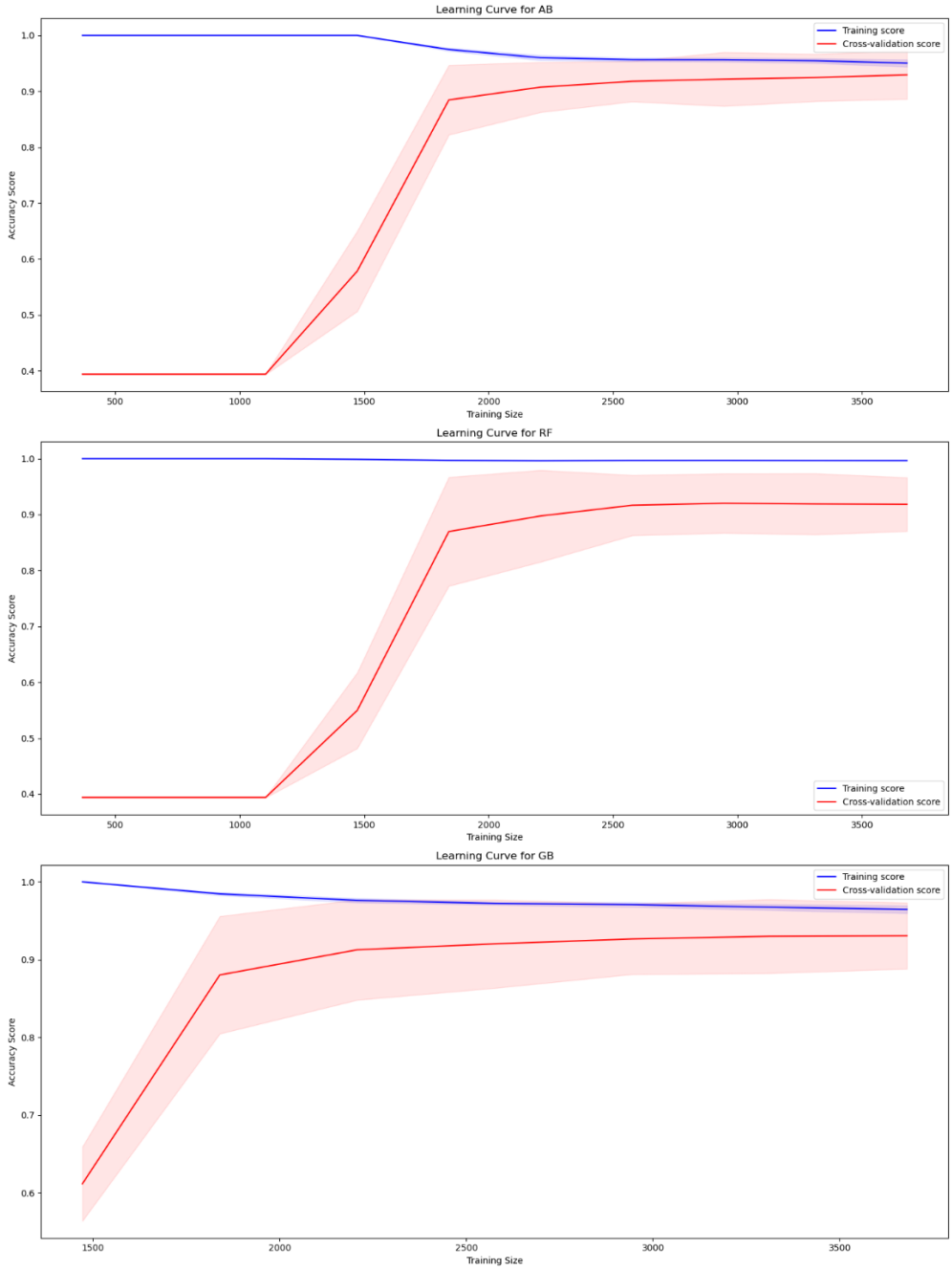


Fig.20: Ensemble models learning curves.

Calibration curves are essential for gauging actual probability scores. The more aligned a model's forecast is to the diagonal line, the better its calibration. Among all ensemble models, the gradient boosting classifier aligns best with this diagonal line. The probability estimate is not very useful for this use case, where we are more interested in the binary classification of the model, measured by accuracy (fig.21).

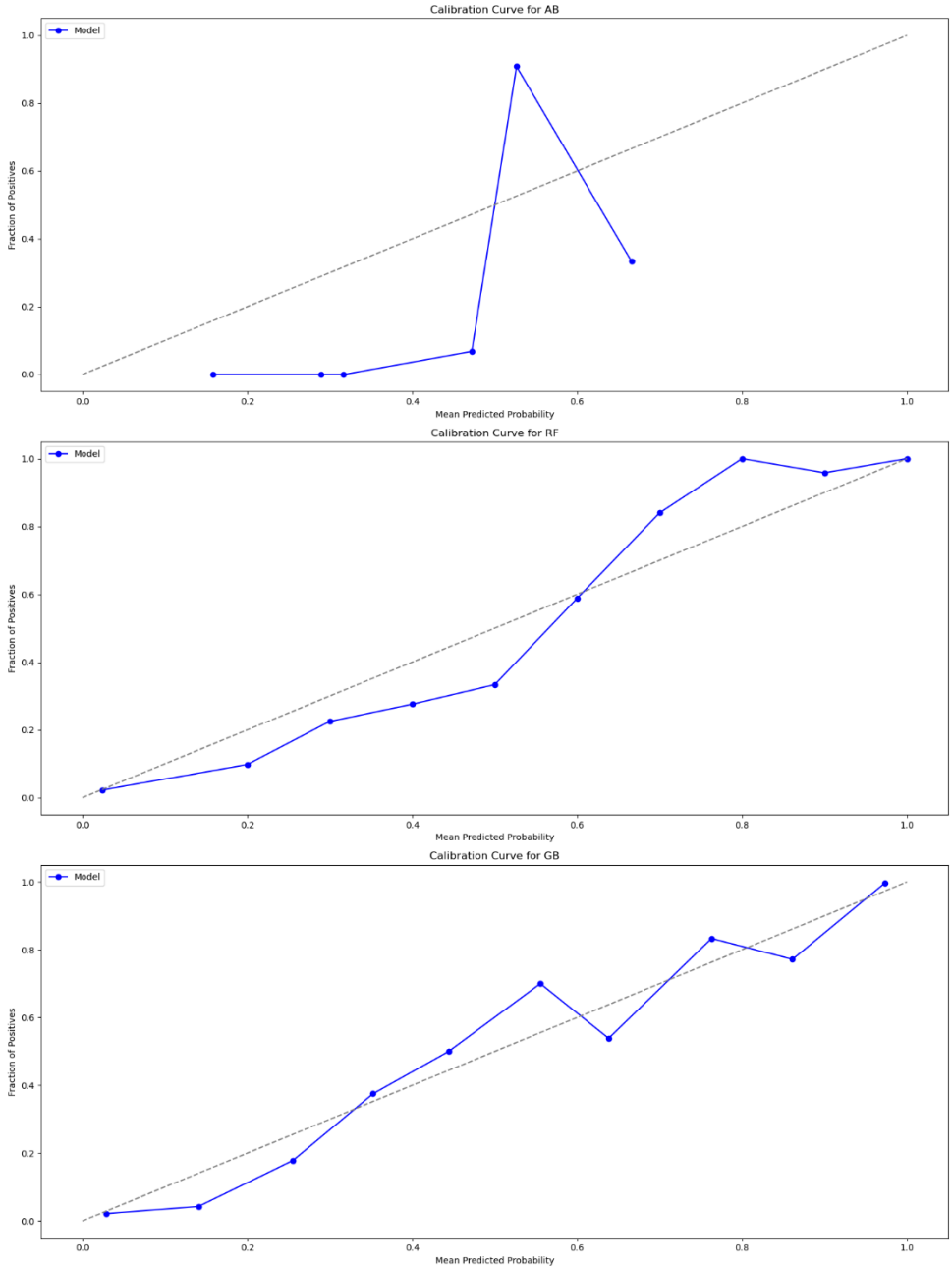


Fig. 21: Ensemble models' calibration curves.

For Stacking an accuracy of 0.9403 was obtained, which is higher than the RF stands at 0.9349. A second hypothesis test was formed to test the performance difference between the stacked model and the best-performing model.

The ROC curve portrays minimal difference between the stacking and gradient boosting classifier (fig.23). Both models perform exceptionally well in reducing type 1 errors, which is of prime importance for this use case.

A paired t-test obtained p-value was 0.3534, which is greater than the common significance level of 0.05. The consequence of this result was the failure to reject the null hypothesis (H0). This means

that there's no statistically significant evidence to suggest differing performance between the stacked model and the RF. The t-statistic of -0.9284 and effect size of 0.0224 backs this decision (fig.22).

Both the gradient boosting and stacked classifiers are well-balanced models (fig.24).

```
Stacking Classifier Accuracy: 0.9402823018458197


Objective: To determine if a stacked model offers superior performance to
the best-performing individual ensemble model in terms of accuracy.


H0: The stacked model's performance is equivalent to the best-performing
individual ensemble model.


H1: The stacked model's performance is significantly better than the
best-performing individual ensemble model.


Best Ensemble Model: RF with Accuracy: 0.9349

Stacked Model vs Best-performing Ensemble Model:
T-statistic: -0.9284
P-value: 0.3534
Effect size:  0.0224
There's no significant performance difference between the stacked model and the RF at the 1.0% level.

Conclusion:
In this analysis, ensemble classifiers and a stacked classifier
was trained. Performances were evaluated and statistically compared.
The results provide insights into whether stacking offers additional
value compared to individual ensemble methods.
```

Fig.22: Hypothesis 3

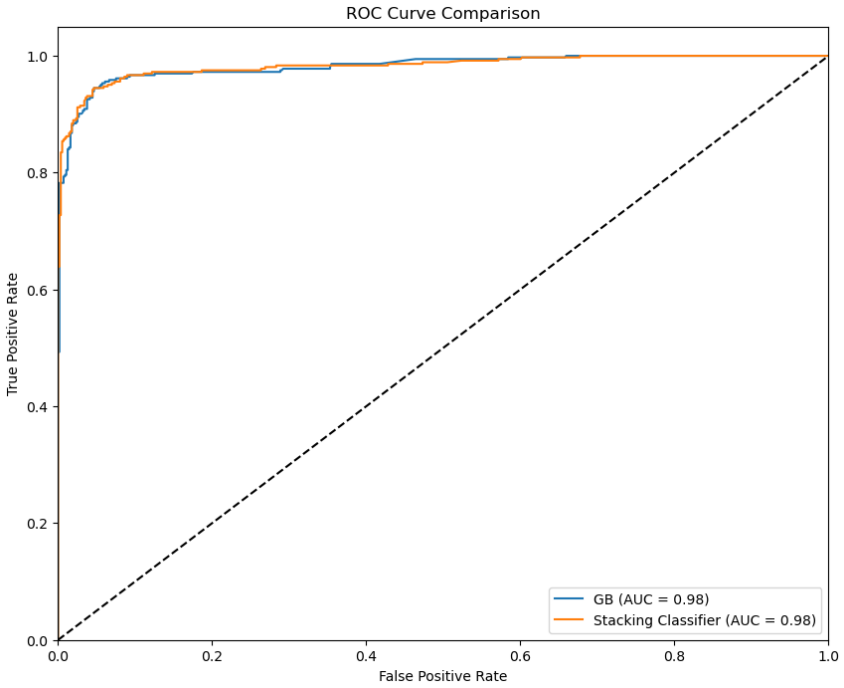


Fig. 23: ROC curves for best classifier vs. stacking classifier.

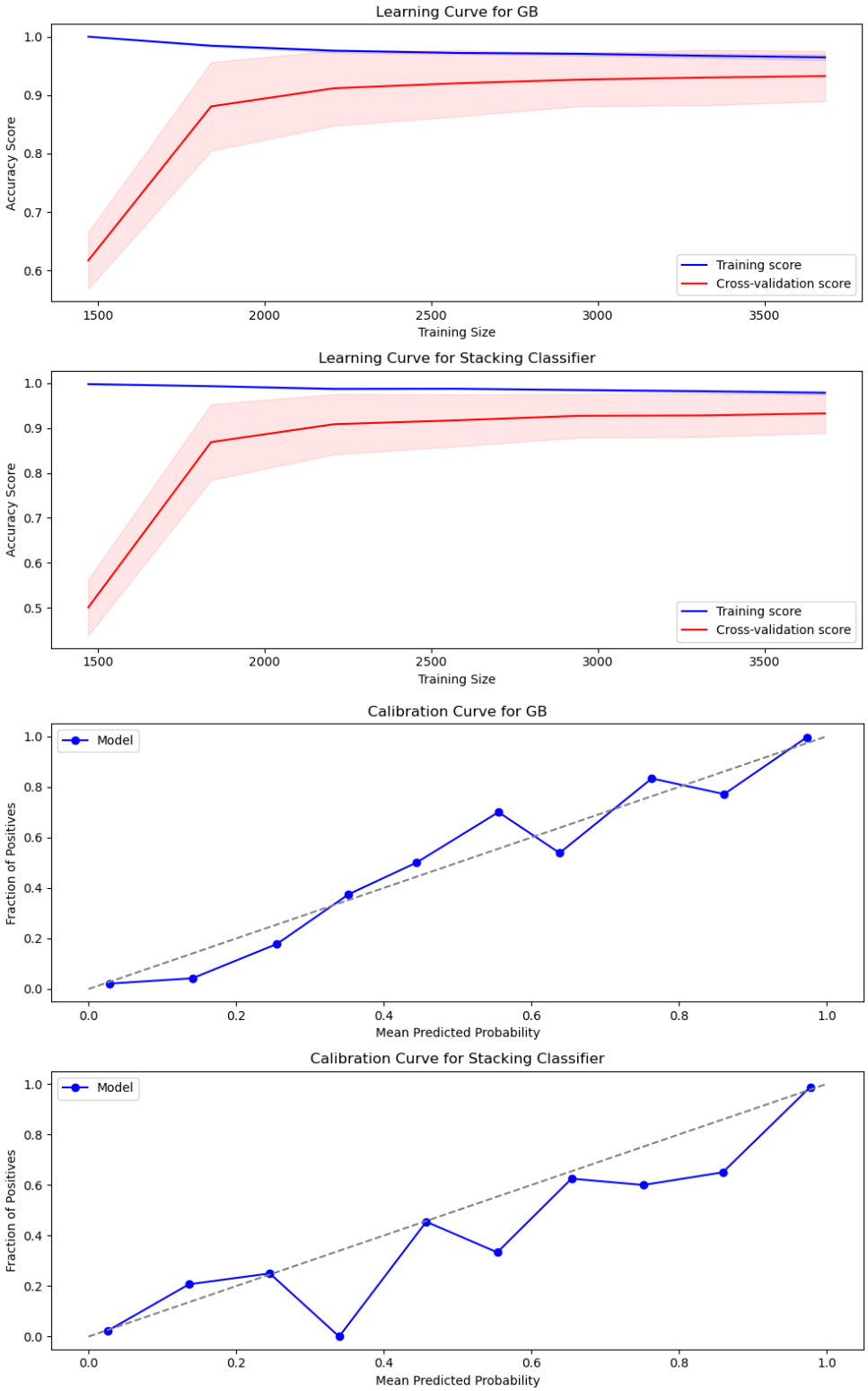


Fig. 24: Stacking vs. Gradient Boosting Curves

```
Stacking Classifier Accuracy: 0.9402823018458197


Objective: To determine if a stacked model offers superior performance to
the best-performing individual ensemble model in terms of accuracy.


H0: The stacked model's performance is equivalent to the best-performing
individual ensemble model.


H1: The stacked model's performance is significantly better than the
best-performing individual ensemble model.


Best Ensemble Model: RF with Accuracy: 0.9349

Stacked Model vs Best-performing Ensemble Model:
T-statistic: -0.9284
P-value: 0.3534
Effect size:  0.0224
There's no significant performance difference between the stacked model and the RF at the 1.0% level.

Conclusion:
In this analysis, ensemble classifiers and a stacked classifier
was trained. Performances were evaluated and statistically compared.
The results provide insights into whether stacking offers additional
value compared to individual ensemble methods.
```

Fig.25: Hypothesis 3

Insights

- Hyper-tuning for KNN did not yield a statistically significant improvement in performance in terms of accuracy.

- All in all, the optimized SVM model performs the best in terms of highest TPR and TNR and lowest FPR and FNR, making it the model of choice for this use case, since avoiding misclassification is a priority.

- Statistical tests reveal no notable differences in performance among all ensemble models studied, but the random forest classifier produces fewer false positives, which is preferable.

- Gradient boosting is the top performer out of the ensemble models with consistent results, RF is similar but with more variation, while Ada boost has good performance and good variability.

- The stacked model has marginally better accuracy than gradient boosting, but the difference isn't statistically significant. Both are well-balanced models and align well with their calibration curves. Choosing between them should extend to aspects like computational efficiency, model complexity, and interpretability.

- Some things which could be changed in using scaled data for the ensemble methods. Reduce the code used for functions by using more in-built methods. Select the training set size

Works Cited

ISLAM, NAIF. “Spambase Dataset.” *Www.kaggle.com*, 21 Jan. 2023, www.kaggle.com/datasets/naifislam/spambase-dataset. Accessed 21 Aug. 2023.

“Multiclass Classification with Xgboost in R.” *Stack Overflow*, 21 Aug. 2021, stackoverflow.com/questions/57824461/multiclass-classification-with-xgboost-in-r. Accessed 21 Aug. 2023.

PABLOV. “Naive Bayes & SVM Spam Filtering.” *Kaggle.com*, 21 Aug. 2015, www.kaggle.com/code/pablovargas/naive-bayes-svm-spam-filtering. Accessed 21 Aug. 2023.

SHARMA, SACHIN. “Spambase Classifier(Acc 95%).” *Kaggle.com*, 21 Aug. 2020, www.kaggle.com/code/sachinsharma1123/spambase-classifier-acc-95. Accessed 21 Aug. 2023.

“Spambase.” *Www.kaggle.com*, 21 Aug. 2018, www.kaggle.com/datasets/monizearabadgi/spambase.

Toushik Wasi, Azmine . “Tabular Playground Series - Aug 2022.” *Kaggle.com*, 21 Aug. 2023, www.kaggle.com/competitions/tabular-playground-series-aug-2022/discussion/341034. Accessed 21 Aug. 2023.

UCI Repository. “UCI Machine Learning Repository.” *Archive.ics.uci.edu*, 30 June 1999, archive.ics.uci.edu/dataset/94/spambase.

DataCamp, DataCamp, 21 Aug. 2023, app.datacamp.com/.

Extra File

Another Python file was created to show some other techniques I learned in this course. Generators, list comprehensions, regular expressions, and web scraping are showcased in this file.

Titles taken from the RTE and BBC news webpages were scraped using the beautiful soup and requests libraries. A function utilizing **kwargs argument was used for this scenario, as after inspecting the HTML elements, RTE and BBC have different naming conventions for their titles.

The generator is commented out, and a list comprehension is used instead because the memory savings from using a generator in this situation would be minimal. The simplicity of using a list

comprehension instead is better, although if you want to scrape thousands of titles, then a generator would be preferable.