Statistics for Data Science
DTSC 620 – M01
Project Assignment 1
Spring 2023
Professor Balagani

Selina Narain
ID 1261565

Understanding our Dataset

To perform classification on a dataset, we need to understand what exactly our dataset is reflecting. Once we understand this, we can classify our data into different classes. This dataset reflects the classification of email messages as spam or ham based on specific words/characters and occurrences.

Within the spam dataset there are:

- 2 classes: Spam OR Ham
- 57 attributes (number of times that certain words or characters occur)
- 4601 instances

<u>Reading Data</u>
Once initializing our dataframe, we can see an overview of what the data represents.

In the following image, we can see that our data has many columns containing the features and each row containing the number of times that word has occurred.

```
#Loading a dataset into a dataframe
spam_dataset_dataframe = pd.read_csv("/Users/selinanarain/Desktop/DTSC 620/ProjectAssignment1/spam.csv")
print('Dataset Loaded...')

spam_dataset_dataframe.head()
```

[319] ✓ 0.3s

```
Dataset Loaded...
```

	make	address	all	3d	our	over	remove	internet	order	mail	...	semicol	paren	bracket	bang	dollar	pound	cap_avg	cap_long	cap_total	Class
0	0.00	0.00	0.29	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.178	0.0	0.044	0.000	0.00	1.666	10	180	ham
1	0.46	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.125	0.0	0.000	0.000	0.00	1.510	10	74	ham
2	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00	0.00	...	0.000	0.000	0.0	0.000	0.000	0.00	1.718	11	55	ham
3	0.33	0.44	0.37	0.0	0.14	0.11	0.00	0.07	0.97	1.16	...	0.006	0.159	0.0	0.069	0.221	0.11	3.426	72	819	spam
4	0.00	2.08	0.00	0.0	3.12	0.00	1.04	0.00	0.00	0.00	...	0.000	0.000	0.0	0.263	0.000	0.00	1.428	4	20	spam

5 rows × 58 columns

The following image shows the describe() method. This method returns descriptive statistics of the data in the DataFrame.

```
spam_dataset_dataframe.describe()
#spam_dataset_dataframe.iloc[0:3]
```

[320] ✓ 0.2s Python

	make	address	all	3d	our	over	remove	internet	order	mail	...	conference	semicol	paren	t
count	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	...	4601.000000	4601.000000	4601.000000	4601.
mean	0.104553	0.213015	0.280656	0.065425	0.312223	0.095901	0.114208	0.105295	0.090067	0.239413	...	0.031869	0.038575	0.139030	0.
std	0.305358	1.290575	0.504143	1.395151	0.672513	0.273824	0.391441	0.401071	0.278616	0.644755	...	0.285735	0.243471	0.270355	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000	0.
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.065000	0.
75%	0.000000	0.000000	0.420000	0.000000	0.380000	0.000000	0.000000	0.000000	0.000000	0.160000	...	0.000000	0.000000	0.188000	0.
max	4.540000	14.280000	5.100000	42.810000	10.000000	5.880000	7.270000	11.110000	5.260000	18.180000	...	10.000000	4.385000	9.752000	4.

8 rows × 57 columns

To further understand our data, we can visualize the dataset features through a histogram. In the following histogram, we can see each feature's plot.

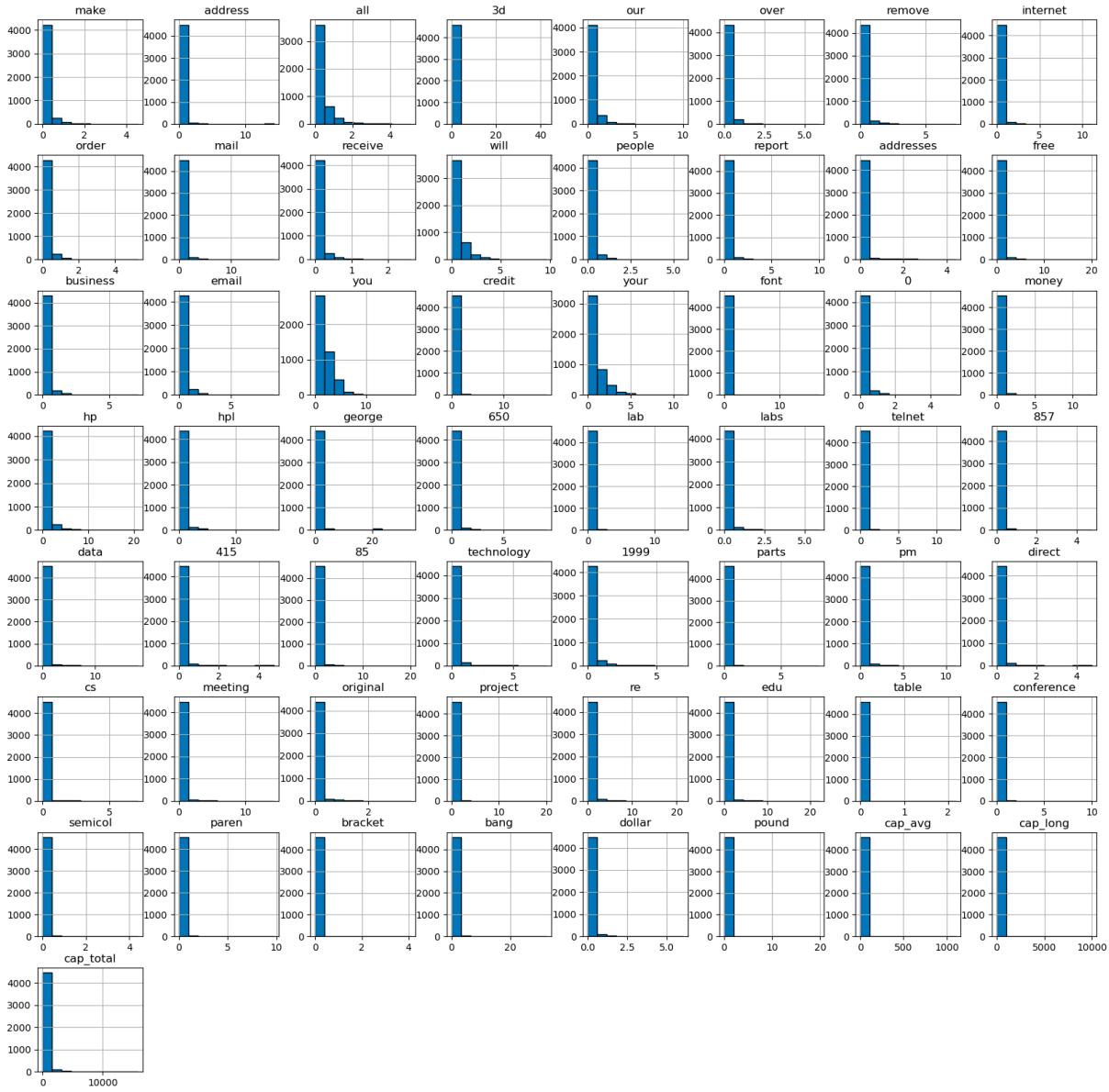


Figure 1: Histogram Plot of Dataset Features

Preparing our Classifier

In order to train our classifier, we will need to prepare our data. In this classifier task, we are testing the classifiers using the first 1000 instances and using the remaining 3601 for training.

Essentially, what is being done in the below image is separating our data into our features and targets. First, we separate the features for the model from the target variable. We initialized an array with all the features from our dataframe. Once doing so, we initialize these feature names to our variable 'X'. After initializing our features, we initialize our target variable 'y' from the 'Class' column in the dataframe since the point of this task is classifying whether these email messages as either spam or ham.

```
# Split dataset in features and target variable
feature_names = ['make', 'address', 'all', '3d', 'our', 'over', 'remove', 'internet', 'order', 'mail',
                 'receive', 'will', 'people', 'report', 'addresses', 'free', 'business', 'email', 'you', 'credit',
                 'your', 'font', '0', 'money', 'hp', 'hpl', 'george', '650', 'lab', 'labs', 'telnet', '857', 'data',
                 '415', '85', 'technology', '1999', 'parts', 'pm', 'direct', 'cs', 'meeting', 'original', 'project',
                 're', 'edu', 'table', 'conference', 'semicol', 'paren', 'bracket', 'bang', 'dollar', 'pound', 'cap_avg',
                 'cap_long', 'cap_total']
X = spam_dataset_dataframe[feature_names] # Features
y = spam_dataset_dataframe.Class # Target variable
[686]  ✓  0.0s
```

Once dealing with our features and targets, we then split our data in our training or testing sets. Using scikit-learn, we can use the train_test_split function to split our features (X) and target (y) even further into training and testing data. In this instance, we can see that our training data is around 80% whereas the test data is about 20%. To view and verify the data has been split, we can view some of the data by using head().

```
# Split df into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1000/4601, random_state=42)
[687]  ✓  0.0s

#View training data
X_train.head()
[688]  ✓  0.0s
```

	make	address	all	3d	our	over	remove	internet	order	mail	...	conference	semicol	paren	bracket	bang	dollar	pound	cap_avg	cap_long	cap_total
2813	0.0	0.0	0.1	0.0	0.0	0.0	0.0	0.1	0.0	0.0	...	0.1	0.074	0.134	0.0	0.000	0.059	0.0	2.529	26	597
2915	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000	0.439	0.0	0.000	0.219	0.0	1.911	11	65
3590	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000	0.037	0.0	0.149	0.000	0.0	10.012	251	791
1552	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000	0.000	0.0	0.344	0.000	0.0	3.250	17	52
2015	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.000	0.000	0.0	0.000	0.000	0.0	1.000	1	5

5 rows × 57 columns

To visualize the splitting of our data, we can also make a scatter plot. As an example, we can use the features ‘make’ and ‘address’ to view and compare the testing and training data.

```
#Scatter plot of training and test data

X_train.plot(kind="scatter", x = "make", y = "address")
X_test.plot(kind="scatter", x = "make", y = "address")
```

[325] ✓ 0.3s

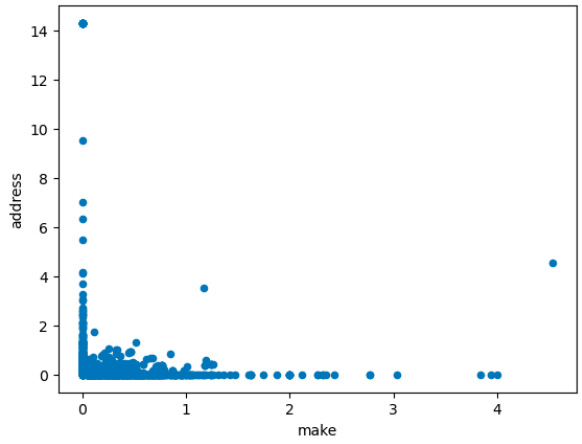
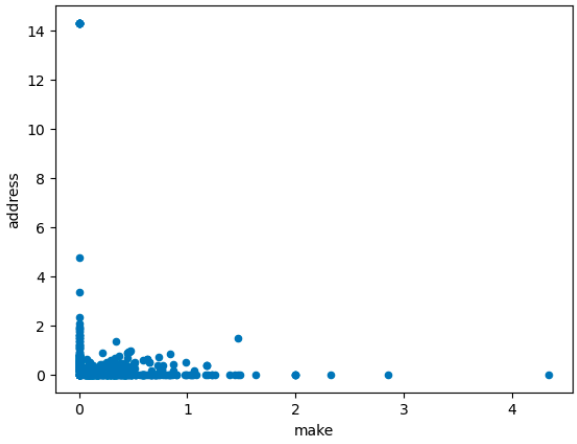
X_train Plot for ‘make’ and ‘address’	X_test Plot for ‘make’ and ‘address’
	

Figure 2: Scatter Plot of Training and Test Data (make, address)

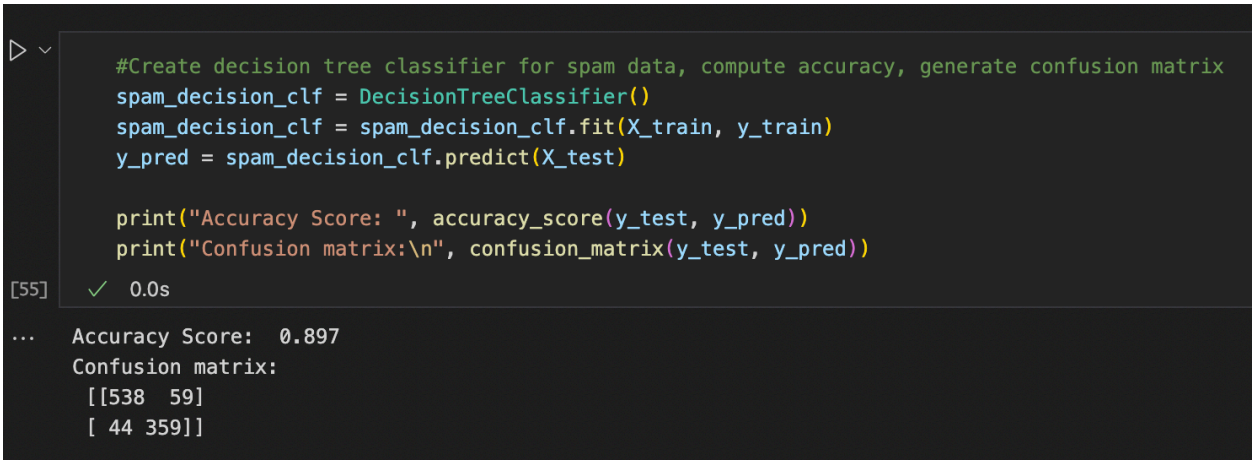
Decision Tree Classifier

Supervised learning is based on a labeled dataset, meaning that the input data is mapped or corresponds to an output variable. A decision tree is a supervised learning algorithm that is used in machine learning. The decision tree algorithm uses specific rules that decides based on either being true or false, similar to how humans arrive to a decision.

To complete the decision tree algorithm, we can use the scikit-learn DecisionTreeClassifier(). First, we initialize the decision tree classifier into an instance 'spam_decision_clf'. Then, we fit the classifier to our training data. After training the model, we can use the model to make predictions on the testing data 'X_test'.

After we perform the decision tree classifier, we can then evaluate the performance of the classifier using accuracy score and confusion matrix. First, we use accuracy score as a metric which views the percentage of the correct predictions made by the model on the testing data. Using scikit-learn we can use 'accuracy_score' to calculate the accuracy score of the classification model. Confusion matrices are used to summarize the classification model showing true positive, true negative, false positive, and false negative predictions. We can use 'confusion_matrix' to view these metrics, mostly focusing on the correct predictions (true positive and true negative).

In the following image below, we created a decision tree model for the spam data and fit the model using our training data we previously split. We then make predications on the new test data that has not been used. Once doing so, we use the accuracy_score and confusion_matrix to evaluate the performance. We can see that the accuracy score is around 89% for our model along with 538 as our true positive and 359 as our true negative.



```
#Create decision tree classifier for spam data, compute accuracy, generate confusion matrix
spam_decision_clf = DecisionTreeClassifier()
spam_decision_clf = spam_decision_clf.fit(X_train, y_train)
y_pred = spam_decision_clf.predict(X_test)

print("Accuracy Score: ", accuracy_score(y_test, y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

[55] ✓ 0.0s

```
Accuracy Score:  0.897
Confusion matrix:
 [[538  59]
 [ 44 359]]
```

For visualization purposes, we can view the decision tree model based on the model we created using graphviz. This decision tree can be seen below.

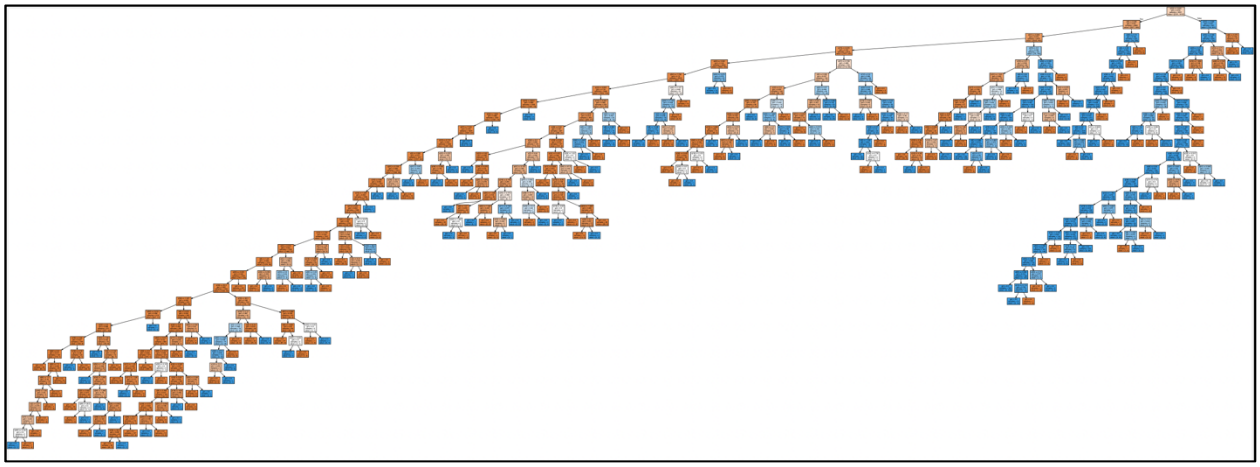


Figure 3: Decision Tree

Decision Tree Conclusions

Based on the performance of the decision tree classifier, we can say that the performance of this classifier is good. The classifier performed at an 89% accuracy which we could say is not too bad, however it can definitely be improved. Based on the confusion matrix, although we have 538 true positives and 359 true negatives, we can still find errors in our classifier from the 59 false positives and 44 false negatives, definitely leaving room for improvement.

Decision Tree Classifier Factoring in Entropy

We can further improve our model by factoring in the impurity. We can measure impurity through the entropy criterion. On our model, we can create another instance specifying the criterion as “entropy”. Similar to what we did before, we use the accuracy_score and confusion_matrix to evaluate the performance. We can see that the accuracy score is around 90% for our model along with 539 as our true positive and 364 as our true negative which is relatively well performed.

```
#Measure the quality of a split using entropy and compare accuracy
spam_decision_clf = DecisionTreeClassifier(criterion = "entropy")
spam_decision_clf = spam_decision_clf.fit(X_train, y_train)

y_pred = spam_decision_clf.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Confusion matrix:\n", confusion_matrix(y_test, y_pred))
```

[57] ✓ 0.1s

```
Accuracy: 0.903
Confusion matrix:
 [[539  58]
 [ 39 364]]
```

Since this model performed better, we can use this for our decision tree classifier and generate a classification report to understand the performance on this model. This report shows:

- Precision: Proportion of true positives (TP) among all positive predictions (TP + FP)
- Recall: Proportion of true positives (TP) among all actual positives (TP + FN)
- F1-score: Mean of precision and recall, providing a balanced measure of both metrics.
- Support: Quantity of samples in each class

```
    #Generate classification report for decision tree classifier considering entropy
    print("Spam Decision Tree Classifier")

    print(classification_report(y_pred, y_test))

    #Based on sklearn ConfusionMatrixDisplay
    cm = confusion_matrix(y_test, y_pred, labels=spam_decision_clf.classes_)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=spam_decision_clf.classes_)

    disp.plot()
    plt.show()
```

[59] ✓ 0.1s

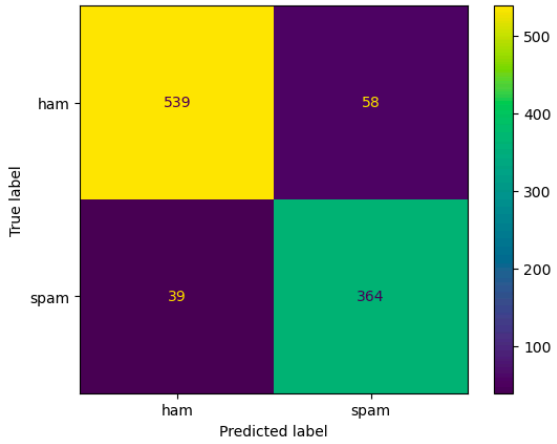
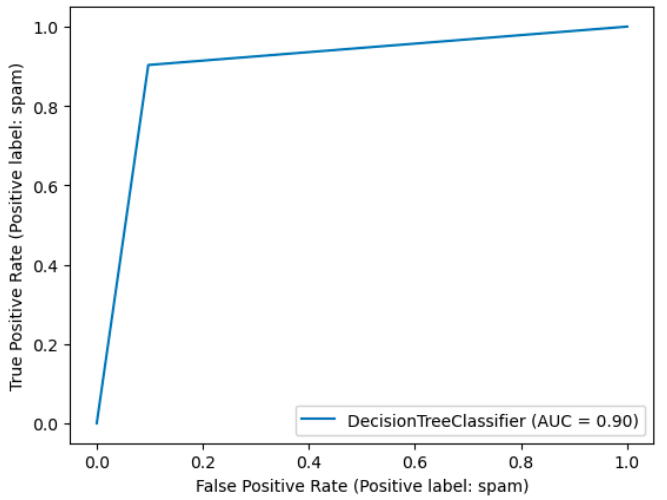
```
Spam Decision Tree Classifier
              precision    recall  f1-score   support

         ham       0.90      0.93      0.92       578
        spam       0.90      0.86      0.88       422

    accuracy                           0.90      1000
   macro avg       0.90      0.90      0.90      1000
weighted avg       0.90      0.90      0.90      1000
```

Based on the classification report, we can see that:

- Precision: Out of all the email messages, 90% predicted messages were correctly classified as ham and 90% were correctly classified as spam.
- Recall: Predicted the model correctly at 93% and 86% for ham and spam respectively.
- F1-score: 92% and 88% for ham and spam respectively.
- Support: 578 classified as ham and 422 classified as spam.

We can also visualize the metrics for the confusion matrix, ROC curve, as well as the decision tree model based on the model we created. This can be seen below.

Confusion Matrix Display	ROC Curve Display:

	
	`RocCurveDisplay.from_estimator(spam_decision_clf, X_test, y_test)`
	`plt.show()`
	[524] ✓ 0.1s

Based on our decision tree model factoring in entropy, the confusion matrix shows: • 539 True Positive (TP) • 58 False Positive (FP) • 39 False Negative (FN) • 364 True Negative (TN)	ROC curve: Used to show the performance of the decision tree based on True Positive Rate and False Positive Rate. Since this classifier shows a curve closer to the top-left corner, it indicates a good performance.

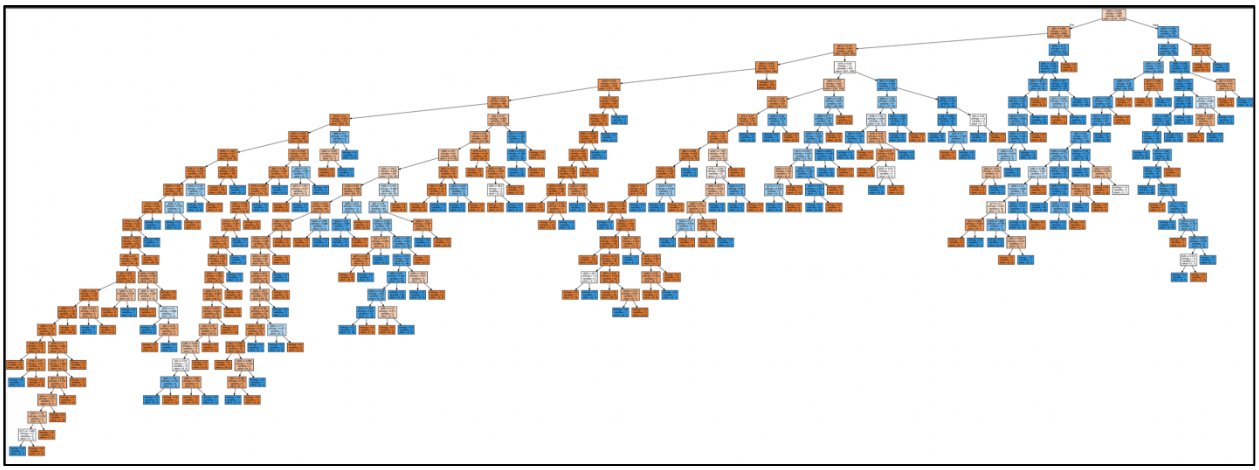


Figure 4: Decision Tree Factoring Entropy

Decision Tree Factoring Entropy Conclusions

Compared to our first decision tree classifier, we can see that the performance of this decision tree when factoring in entropy performs slightly better than the default classifier. In this classifier, it has performed with a 90% accuracy, which is an improvement of 1% from the previous decision tree classifier. We can also see an improvement from the confusion matrix. We can see an improvement of in the true positives which has increased from 538 to 539 as well as the true negatives which increased from 359 to 364. We can also see improvement as the false positives decreased from 59 to 58 and false negatives decreased from 44 to 39. This shows that the misclassification rate is lower than prior when including the entropy parameter.

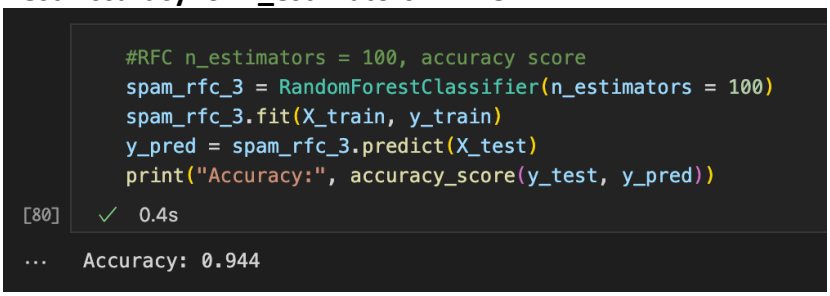
Random Forest Classifier

Random forest is a supervised learning algorithm that is used in machine learning. This algorithm combines decision trees which have been trained on a random subset of training data and a random subset of input features. This algorithm is used mostly to avoid overfitting and improve performance of a model in a more generalized way by calculating the average of the prediction of individual trees.

As a part of our task, the random forest classifier accuracies were compared for base learners of 10, 50, 100, 500, 1000, and 5000 along with considering the various number of features such as auto, sqrt, log2.

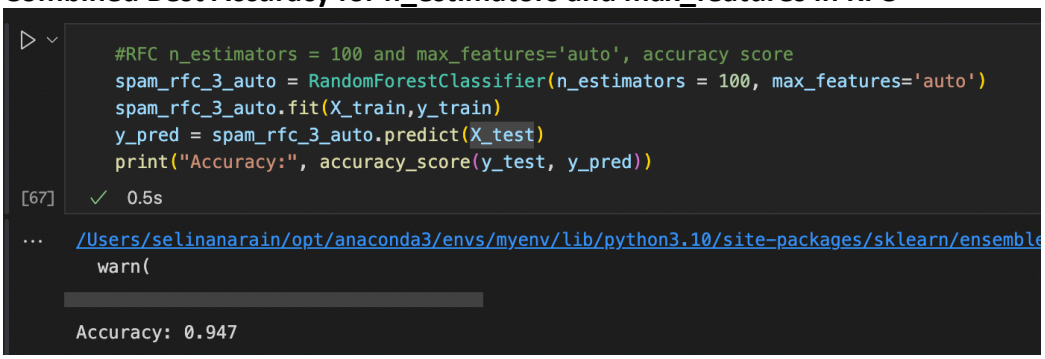
Different random forest classifiers were performed for different n_estimators. We performed the classifier for n_estimators for 10, 50, 100, 500, 1000, and 5000. When performing these classifiers and viewing the corresponding accuracy scores, I was able to conclude n_estimators=100 has the best accuracy score of around 94% in comparison to all the other classifiers performed. This can be viewed in the below image.

Best Accuracy for n_estimators in RFC



Once we have the best accuracy for n_estimators, we can then move on to consider the best parameter that fits for the number of features such as auto, sqrt, and log2. When performing each classifier using n_estimators=100 (best performed n_ estimators) and auto, sqrt, and log2, we can find that max_features best perform at 'auto'. By combining n_estimators and max_features in our random forest classifier, we can conclude that these are the best parameters to perform that most accurate classification.

Combined Best Accuracy for n_estimators and max_features in RFC



We can also generate a classification report similar to the decision tree to understand the performance on this model.

Random Forest Classification Report

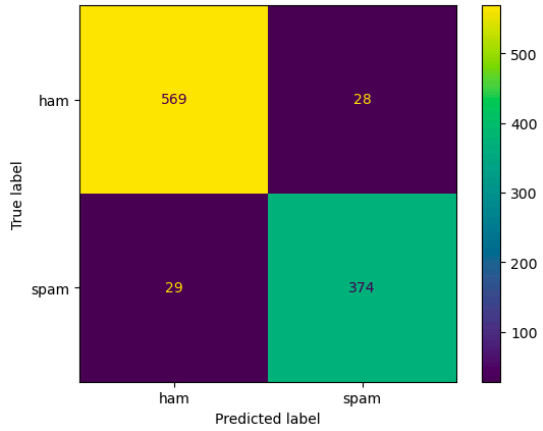
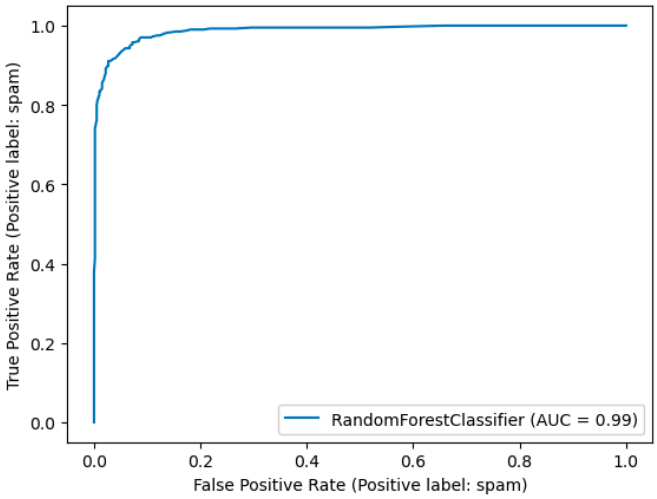
```
#Generate classification report for random forest classifier considering entropy
print("Spam Random Forest Classifier")

print(classification_report(y_pred, y_test))

cm = confusion_matrix(y_test, y_pred, labels=best_spam_rfc.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=best_spam_rfc.classes_)

disp.plot()
plt.show()
```

[71] ✓ 0.1s

```
Spam Random Forest Classifier
              precision    recall  f1-score   support

         ham       0.95      0.95      0.95       598
        spam       0.93      0.93      0.93       402

    accuracy                           0.94      1000
   macro avg       0.94      0.94      0.94      1000
weighted avg       0.94      0.94      0.94      1000
```

Based on the classification report, we can see that:

- Precision: Out of all the email messages, 95% predicted messages were correctly classified as ham and 93% were correctly classified as spam.
- Recall: Predicted the model correctly at 95% and 93% for ham and spam respectively.
- F1-score: 95% and 93% for ham and spam respectively.
- Support: 578 classified as ham and 402 classified as spam.

Confusion Matrix Display	ROC Curve Display:
	 
Based on our decision tree model factoring in entropy, the confusion matrix shows: • 569 True Positive (TP) • 28 False Positive (FP) • 29 False Negative (FN) • 374 True Negative (TN)	ROC curve: Used to show the performance of the decision tree based on True Positive Rate and False Positive Rate. Since this classifier shows a curve closer to the top-left corner, it indicates a better performance as it is closer to 1.

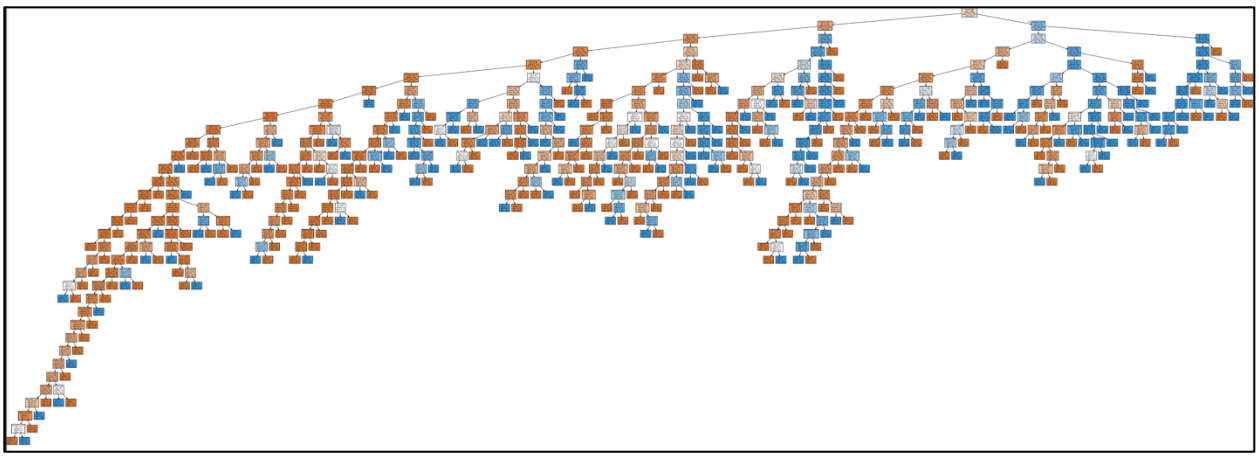


Figure 5: Visualize Individual Decision Trees in Best Random Forest Classifier

Random Forest Classifier Conclusions

When performing the random forest classifier, we can see that it performed significantly better than the decision tree classifier. From this classifier, we tested various parameters for the classifier to better improve accuracy. Once testing, we found that the best parameters that resulted in the best accuracy. These parameters ended up being n_estimators = 100 and max_features = 'auto' in the random forest classifier. The accuracy for the random forest classifier with these parameters turned out to be about 94.7%, which is slightly better accuracy than just having n_estimators = 100 which has an accuracy of 94.4%. Our classifier improved by specifying certain parameters to find what best suites it.

Overall Conclusion

Based on conducting the decision tree classifiers and the random forest classifiers, one of the main takeaways found is that finding the best parameters for our classifiers can improve the accuracy of our performance. In the decision tree classifier, we can see that specifying the criterion parameter caused our classifier to improve in performance. Similarly in the random forest classifier, we can see that by specifying the n_estimators (aggregation of specific number of samples) as well as max_features (number of features to consider at each split), we can see that the random forest performed much better than default.