Statistics for Data Science
DTSC 620 – M01
Project Assignment 2
Spring 2023
Professor Balagani

Selina Narain
ID 1261565

Understanding our Dataset

To perform various models on the dataset, there is need to understand what our dataset is reflecting. 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>

First, we initialize our data frame which can be seen below.

```
#Loading a dataset into a dataframe
spam_dataset_dataframe = pd.read_csv("/Users/selinanarain/Desktop/DTSC 620/ProjectAssignment2/spam.csv")
print('Dataset Loaded...')
```

[35] ✓ 0.0s

```
Dataset Loaded...
```

Then, from the data frame we can look through the data to see if these are any missing values. These values can cause a skew of data. However, from the image below we can see that there are no missing values.

```
#Check for missing values
spam_dataset_dataframe.isna().sum()
```

[3] ✓ 0.0s

```
Output exceeds the size limit. Open the full output data in a text editor
make          0
address       0
all           0
3d            0
our           0
over          0
remove        0
internet      0
order         0
mail          0
receive       0
will          0
people        0
report        0
addresses     0
free          0
business      0
email         0
you           0
credit        0
your          0
font          0
0             0
money         0
hp            0
...
cap_avg       0
cap_long      0
cap_total     0
Class         0
dtype: int64
```

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

```
#Describe dataset
spam_dataset_dataframe.describe()
```

[37] ✓ 0.2s Python

	make	address	all	3d	our	over	remove	internet	order	mail	...	conference	sen
count	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	4601.000000	...	4601.000000	4601.00(
mean	0.104553	0.213015	0.280656	0.065425	0.312223	0.095901	0.114208	0.105295	0.090067	0.239413	...	0.031869	0.03
std	0.305358	1.290575	0.504143	1.395151	0.672513	0.273824	0.391441	0.401071	0.278616	0.644755	...	0.285735	0.24
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.00(
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.00(
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.00(
75%	0.000000	0.000000	0.420000	0.000000	0.380000	0.000000	0.000000	0.000000	0.000000	0.160000	...	0.000000	0.00(
max	4.540000	14.280000	5.100000	42.810000	10.000000	5.880000	7.270000	11.110000	5.260000	18.180000	...	10.000000	4.38

8 rows × 57 columns

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.

```
spam_dataset_dataframe.head()
```

[38] ✓ 0.0s Python

	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

Preparing our Models

In order to train our models, we will need to prepare our data.

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 data frame. 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 data frame since the point of this task is classifying whether these email messages as either spam or ham.

```
#Split dataset into 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
```

[5] ✓ 0.0s

Voting Classifier

Our first task is to fuse three classifiers into one. Once dealing with the features and target variable, we then split our data in our training and 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. We are tasked with testing with 1000/4601 and training with the remaining. This can be seen in the below image.

Fused Model - Voting Classifier using 1000/4601 test size

```
#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)
```

[59] ✓ 0.0s

A voting classifier is an ensemble machine learning model that combines the predictions of multiple individual classifiers to make a final prediction. It is a type of ensemble learning method that improves the overall predictive performance through these votes.

Once splitting the data, we can start to fuse the classifiers. To do so, we use the majority voting rule, using a voting classifier. We are tasked to fuse the Decision Tree Classifier, Gaussian Naïve Bayes Classifier, and Logistic Regression Classifier. As we can see from the image below, we can do so by defining each of the models themselves.

```
#Define models for voting classifier
spam_dt = DecisionTreeClassifier()
spam_nb = GaussianNB()
spam_lr = LogisticRegression()
```

[60] ✓ 0.0s

Once we define the models we are using, we can then specify them within the VotingClassifier() parameters. Once the voting classifier is defined with the specified classifiers, we can then fit and train the voting classifier model.

```
#Voting Classifier Model
spam_vc = VotingClassifier(estimators=[('dt', spam_dt), ('gnb', spam_nb), ('lr', spam_lr)], voting='hard')

#Train the model
spam_vc1 = spam_vc.fit(X_train, y_train)

#Predict on testing data
y_pred_vc = spam_vc1.predict(X_test)
```

[90] ✓ 0.1s

After we perform the voting 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).

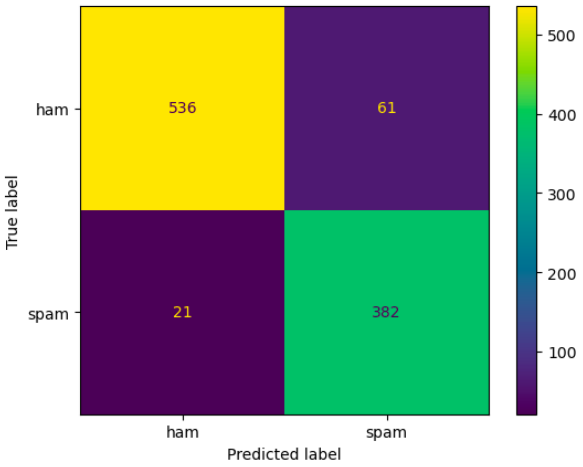
In the following image below, we can use the accuracy_score and confusion_matrix to evaluate the performance. We can also use classification report as well to view the precision, recall, f1-score, and support.

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

print(classification_report(y_pred_vc, y_test))

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

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

[91] ✓ 0.1s

```
Accuracy: 0.918
Confusion matrix:
 [[536  61]
 [ 21 382]]
              precision    recall  f1-score   support

         ham       0.90      0.96      0.93       557
        spam       0.95      0.86      0.90       443

    accuracy                           0.92      1000
   macro avg       0.92      0.91      0.92      1000
weighted avg       0.92      0.92      0.92      1000
```

Confusion Matrix Display for Voting Classifier



Voting Classifier Conclusions

We can see that the accuracy score is around 91% for our model along with 536 as our true positive and 382 as our true negative. This is relatively good since the accuracy comes out to over 90%

AdaBoost Ensemble with Decision Tree

Our second task is to compare the accuracies of the fused model with AdaBoost Ensemble with Decision Tree as the base learner. Since we already dealt with the features and target variable, we can split our data into our training and testing sets. We are tasked with testing with 1000/4601 and training with the remaining for the model. This is shown in the image below.

```
[15]  #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)
      ✓ 0.0s
```

AdaBoost is a boosting ensemble learning model. Essentially, this model boosts how it learns by learning from the previous mistakes. This model also works well with the decision tree classifier because the model makes predictions by calculating the weighted average of weak classifiers.

Once splitting the data, we can start to build our model. To do so, we use AdaBoostClassifier() to create the model. When creating the model, we also define the base estimator to be the have decision tree as the base learner. Once the classifier is defined with the specified base learner, we then fit and train the AdaBoost model.

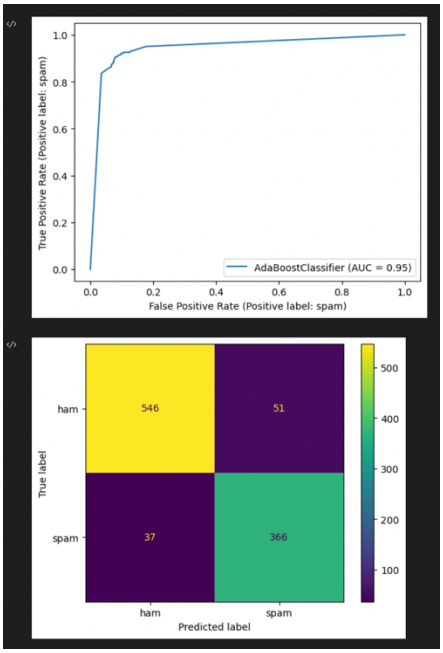
```
[16]  # Create adaboost classifer object
      spam_abc = AdaBoostClassifier(base_estimator=spam_dt)

      # Train Adaboost Classifer
      spam_abc = spam_abc.fit(X_train, y_train)

      #Predict the response for test dataset
      y_pred_abc = spam_abc.predict(X_test)
      ✓ 2.3s
```

In the following image below, we can use the accuracy_score and confusion_matrix to evaluate the performance. We can also use classification report as well to view the precision, recall, f1-score, and support. We can see that the accuracy score is around 91% for our model along with 546 as our true positive and 366 as our true negative.

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

print(classification_report(y_pred_abc, y_test))

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

#Plot ROC curve
RocCurveDisplay.from_estimator(spam_abc, X_test, y_test)

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

[94] ✓ 0.1s

```
Accuracy: 0.912
Confusion matrix:
 [[546  51]
 [ 37 366]]
              precision    recall  f1-score   support

         ham       0.91      0.94      0.93       583
        spam       0.91      0.88      0.89       417

    accuracy                           0.91      1000
   macro avg       0.91      0.91      0.91      1000
weighted avg       0.91      0.91      0.91      1000
```



AdaBoost Ensemble with Decision Tree Conclusions

Based on the accuracy of the AdaBoost classifier, we can see that both the voting classifier and AdaBoost have around the same accuracies of 91%. Also, the confusion matrices reflect the true positives as 546 as our true positive and 366 as our true negative as opposed to the voting classifier which is 536 as our true positive and 382 as our true negative.

Random Forest Classifier

Our third task is to compare the accuracies of the fused model with a Random Forest classifier. As we did for the other models, we split our data into our training and testing sets. We are tasked with testing with 1000/4601 and training with the remaining for the model. Once splitting the data, we build the model using RandomForestClassifier() to create the model. We then fit and train the Random Forest model.

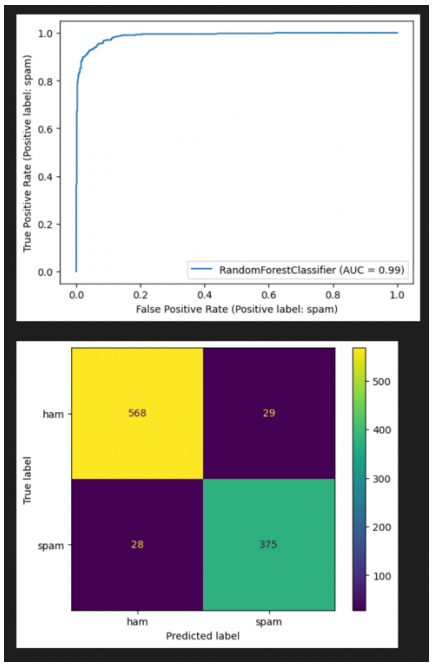
```
#Split df into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1000/4601, random_state=42)
```
[95] ✓ 0.0s

```
# Create Random Forest Classifer model
spam_rfc = RandomForestClassifier(n_estimators = 1000)

#Train the model
spam_rfc = spam_rfc.fit(X_train, y_train)

#Predict on testing data
y_pred_rfc = spam_rfc.predict(X_test)
```
[96] ✓ 4.2s

We can see from the below image that the accuracy score is around 94.3% for our model along with 568 as our true positive and 375 as our true negative.

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

print(classification_report(y_pred_rfc, y_test))

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

#Plot ROC curve
RocCurveDisplay.from_estimator(spam_rfc, X_test, y_test)

disp.plot()
plt.show()
```
[97] ✓ 0.3s

```
Accuracy: 0.943
Confusion matrix:
 [[568  29]
 [ 28 375]]
              precision    recall  f1-score   support

         ham       0.95      0.95      0.95       596
        spam       0.93      0.93      0.93       404

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



Random Forest Conclusions

Overall, we can see this model performed at an accuracy of 94.3%. This is quite higher than the accuracy of the AdaBoost and Fused models at 91.2% and 91.8% respectively. We can also see the ROC curve has an AUC of 0.99 which is quite close to 1, being near perfect.

Impact of Training Sample Size

We can also investigate the influence of training sample size on the accuracies of the fused classifier as well as the AdaBoost Ensemble with Decision Tree as the base learner. We can compare their accuracies using different training-test splits: 50%-50%, 60%-40%, 70%-30%, and 80%-20%. By examining the impact of varying training sample sizes, we can gain insights into the performance of these models under different scenarios. The results will also provide conclusions about the effectiveness of these classifiers and the relationship between training sample size and predictive accuracy.

Voting Classifier Model

50%-50%	50% Train – 50% Test: Accuracy = 0.9182963928726641 Confusion Matrix: [[1261 140] [48 852]] Classification Report: precision recall f1-score support ham 0.96 0.90 0.93 1401 spam 0.86 0.95 0.90 900 accuracy 0.92 2301 macro avg 0.91 0.92 0.92 2301 weighted avg 0.92 0.92 0.92 2301	91.8% accuracy 1261 true positive, 852 true negative
60%-40%	60% Train – 40% Test: Accuracy = 0.9266702878870179 Confusion Matrix: [[1035 99] [36 671]] Classification Report: precision recall f1-score support ham 0.97 0.91 0.94 1134 spam 0.87 0.95 0.91 707 accuracy 0.93 1841 macro avg 0.92 0.93 0.92 1841 weighted avg 0.93 0.93 0.93 1841	92.6% accuracy 1035 true positive, 671 true negative
70%-30%	70% Train – 30% Test: Accuracy = 0.9268645908761767 Confusion Matrix: [[761 76] [25 519]] Classification Report: precision recall f1-score support ham 0.97 0.91 0.94 837 spam 0.87 0.95 0.91 544 accuracy 0.93 1381 macro avg 0.92 0.93 0.92 1381 weighted avg 0.93 0.93 0.93 1381	92.6% accuracy 761 true positive, 519 true negative
80%-20%	80% Train – 19% Test: Accuracy = 0.9120521172638436 Confusion Matrix: [[493 62] [19 347]] Classification Report: precision recall f1-score support ham 0.96 0.89 0.92 555 spam 0.85 0.95 0.90 366 accuracy 0.91 921 macro avg 0.91 0.92 0.91 921 weighted avg 0.92 0.91 0.91 921	91.2% accuracy 493 true positive, 347 true negative

AdaBoost Ensemble with Decision Tree

50%-50%	50% Train – 50% Test: Accuracy = 0.8970013037809648 Confusion Matrix: [[1262 139] [98 802]] Classification Report: precision recall f1-score support ham 0.93 0.90 0.91 1401 spam 0.85 0.89 0.87 900 accuracy 0.90 2301 macro avg 0.89 0.90 0.89 2301 weighted avg 0.90 0.90 0.90 2301	89.7% accuracy 1262 true positive, 802 true negative
60%-40%	60% Train – 40% Test: Accuracy = 0.8967952199891364 Confusion Matrix: [[1016 118] [72 635]] Classification Report: precision recall f1-score support ham 0.93 0.90 0.91 1134 spam 0.84 0.90 0.87 707 accuracy 0.90 1841 macro avg 0.89 0.90 0.89 1841 weighted avg 0.90 0.90 0.90 1841	89.7% accuracy 1016 true positive, 635 true negative
70%-30%	70% Train – 30% Test: Accuracy = 0.9029688631426502 Confusion Matrix: [[759 78] [56 488]] Classification Report: precision recall f1-score support ham 0.93 0.91 0.92 837 spam 0.86 0.90 0.88 544 accuracy 0.90 1381 macro avg 0.90 0.90 0.90 1381 weighted avg 0.90 0.90 0.90 1381	90.3% accuracy 759 true positive, 488 true negative
80%-20%	80% Train – 19% Test: Accuracy = 0.9077090119435396 Confusion Matrix: [[510 45] [40 326]] Classification Report: precision recall f1-score support ham 0.93 0.92 0.92 555 spam 0.88 0.89 0.88 366 accuracy 0.91 921 macro avg 0.90 0.90 0.90 921 weighted avg 0.91 0.91 0.91 921	90.8% accuracy 510 true positive, 326 true negative

Training Sample Size Conclusions

Based on the voting classifier test train split accuracies, they are all around 91-92%. However, 60-40 and 70-30 have the same accuracies. Also, the precision is more accurate for spam as opposed to ham. However, the voting classifier performs well in terms of classifying the data based on majority in votes.

Based on the AdaBoost ensemble test train split accuracies, they increase when the training sizes increases. We can see that the best accuracy comes from the 80-20 split with 90.8%. As opposed to voting classifier, we can see the AdaBoost classifier performs a bit poorer than the voting classifier.