Spam Detector¶

In [41]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

Retrieve the Data¶

The data is located at https://static.bc-edx.com/mbc/ai/m4/datasets/spam-data.csv

Dataset Source: UCI Machine Learning Library

Import the data using Pandas. Display the resulting DataFrame to confirm the import was successful.

In [42]:

```
# Import the data
data = pd.read_csv("https://static.bc-edx.com/mbc/ai/m4/datasets/spam-data.csv")
data.head()
```

Out[42]:

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_remove	word_freq_internet	word_freq_order	word_freq_ma
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63

5 rows × 58 columns

Predict Model Performance¶

You will be creating and comparing two models on this data: a Logistic Regression, and a Random Forests Classifier. Before you create, fit, and score the models, make a prediction as to which model you think will perform better. You do not need to be correct!

Write down your prediction in the designated cells in your Jupyter Notebook, and provide justification for your educated guess.

My assumption would be that the Random Forest model will be more accurate at predicting spam. My reasoning behind this is that the dataset has a lot of underlying points that I visually can see as decision nodes that if incorporated into a Random Forest could improve and strengthen the performance of the model.

Split the Data into Training and Testing Sets¶

In [43]:

```
# Create the labels set `y` and features DataFrame `X`
# Make copy of the dataset
X = data.copy()
# Drop the target column
X.drop("spam", axis = 1, inplace = True)

# Define the target set
y = data["spam"]
```

In [44]:

```
# Check the balance of the labels variable (`y`) by using the `value_counts` function.
y.value_counts()
```

Out[44]:

```
0    2788
1    1813
Name: spam, dtype: int64
```

In [45]:

```
# Split the data into X_train, X_test, y_train, y_test
X_train, X_test, y_train, y_test = train_test_split(X,y)
```

Scale the Features¶

Use the `StandardScaler` to scale the features data. Remember that only `X_train` and `X_test` DataFrames should be scaled.

In [46]:

```
from sklearn.preprocessing import StandardScaler

# Create the StandardScaler instance
scaler = StandardScaler()
```

In [47]:

```
# Fit the Standard Scaler with the training data
X_scaler = scaler.fit(X_train)
```

In [48]:

```
# Scale the training data
X_train_scaled = X_scaler.transform(X_train)
```

```
X_test_scaled = X_scaler.transform(X_test)
```

Create and Fit a Logistic Regression Model¶

Create a Logistic Regression model, fit it to the training data, make predictions with the testing data, and print the model's accuracy score. You may choose any starting settings you like.

In [49]:

```
# Train a Logistic Regression model and print the model score
from sklearn.linear_model import LogisticRegression

# Create the LogisticRegression function and assign it to a variable
logistic_regression_model = LogisticRegression(random_state = 1)

# Fit the model
logistic_regression_model.fit(X_train_scaled, y_train)

# Score the model
print(f"Training Data Score: {logistic_regression_model.score(X_train_scaled, y_train)}")
print(f"Testing Data Score: {logistic_regression_model.score(X_test_scaled, y_test)}")
```

```
Training Data Score: 0.927536231884058
Testing Data Score: 0.9287576020851434
```

In [50]:

```
# Make and save testing predictions with the saved logistic regression model using the test data
predictions = logistic_regression_model.predict(X_train_scaled)

# Review the predictions
results_df = pd.DataFrame({"Prediction": predictions,
                           "Actual": y_train})
results_df
```

Out[50]:

	Prediction	Actual
2283	0	0
1670	1	1
514	1	1
2253	0	0
2918	1	0
...	...	...
417	1	1
3613	0	0
2364	0	0
446	1	1
3715	0	0

3450 rows × 2 columns

In [51]:

```
# Calculate the accuracy score by evaluating `y_test` vs. `testing_predictions`.
testing_predictions = logistic_regression_model.predict(X_test_scaled)

# Save both the test predictions and actual test values to a dataframe
results_df = pd.DataFrame({"Testing Data Predictions": testing_predictions,
                           "Testing Data Actual Targets": y_test})

# Display the dataframe
results_df

# Accuracy Score
acc_score = accuracy_score(y_test, testing_predictions)

# Display accuracy score
print(f"Accuracy Score: {acc_score}")
```

```
Accuracy Score: 0.9287576020851434
```

Create and Fit a Random Forest Classifier Model¶

Create a Random Forest Classifier model, fit it to the training data, make predictions with the testing data, and print the model's accuracy score. You may choose any starting settings you like.

In [55]:

```
# Train a Random Forest Classifier model and print the model score
from sklearn.ensemble import RandomForestClassifier

# Create the random forest classifier instance
rf_model = RandomForestClassifier(n_estimators = 128, random_state = 1)

# Fit the model
rf_model = rf_model.fit(X_train_scaled, y_train)

# Print score
print(f"Training Data Score: {rf_model.score(X_train_scaled, y_train)}")
print(f"Testing Data Score: {rf_model.score(X_test_scaled, y_test)}")
```

```
Training Data Score: 0.9994202898550725
Testing Data Score: 0.9600347523892268
```

In [56]:

```
# Make and save testing predictions with the saved logistic regression model using the test data
predictions = rf_model.predict(X_test_scaled)

# Make dataframe with predictions
results_df = pd.DataFrame({"Predictions": predictions,
                           "Actuals": y_test})
# Review the predictions
results_df
```

Out[56]:

	Predictions	Actuals
3355	0	0
4102	0	0
3927	0	0
3271	0	0
1934	0	0
...	...	...
1115	1	1
2130	0	0
1520	1	1
3533	0	0
3317	0	0

1151 rows × 2 columns

In [57]:

```
# Calculate the accuracy score by evaluating `y_test` vs. `testing_predictions`.
acc_score = accuracy_score(y_test, predictions)

# Print accuracy score
print(f"Accuracy Score: {acc_score}")
```

```
Accuracy Score: 0.9600347523892268
```

Evaluate the Models¶

Which model performed better? How does that compare to your prediction? Write down your results and thoughts in the following markdown cell.

The RandomForest model had an accuracy score of 0.960 while the LogisticRegression model had an accuracy score of 0.928 thus showing that the RandomForest model performed better. This validates my initial assumption at the beginning of this project and in theory makes sense as RandomForests combine many decision tree models to improve the prediction on the dataset.

In [ ]: