Decision Tree vs Random Forest

In this project, spam emails from the UCR email spam dataset (https://archive.ics.uci.ed/ml/datasets/Spambase) is used to compare Decision Tree and Random Forest models.

Load Packages

In [28]:

```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
np.random.seed(1)

import pandas as pd
pd.set_option('display.max_columns', None)

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import classification_report
from sklearn.pipeline import Pipeline

import pydotplus
from IPython.display import Image
```

Data Exploration

In [2]:

```
df = pd.read_csv("spambase.data",header=None)
```

The dataset has been pre-engineereed and features have been extracted according to spambase.DOCUMENTATION file in the downloaded file:

- The last column of 'spambase.data' denotes whether the e-mail was considered spam (1) or not (0).
- 48 continuous real [0,100] attributes of type word_freq_WORD = percentage of words in the e-mail that match WORD, i.e. 100 * (number of times the WORD appears in the e-mail) / total number of words in e-mail.

- 6 continuous real [0,100] attributes of type char_freq_CHAR = percentage of characters in the e-mail that match CHAR, i.e. 100 * (number of CHAR occurences) / total characters in e-mail
- 1 continuous real [1,...] attribute of type capital_run_length_average = average length of uninterrupted sequences of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_longest = length of longest uninterrupted sequence of capital letters
- 1 continuous integer [1,...] attribute of type capital_run_length_total = sum of length of uninterrupted sequences of capital letters = total number of capital letters in the e-mail
- 1 nominal {0,1} class attribute of type spam = denotes whether the e-mail was considered spam (1) or not (0), i.e. unsolicited commercial e-mail.
- Missing Attribute Values: None
- Class Distribution: Spam 1813 (39.4%) Non-Spam 2788 (60.6%)

In [12]:
```
df.head(3)
```

Out[12]:

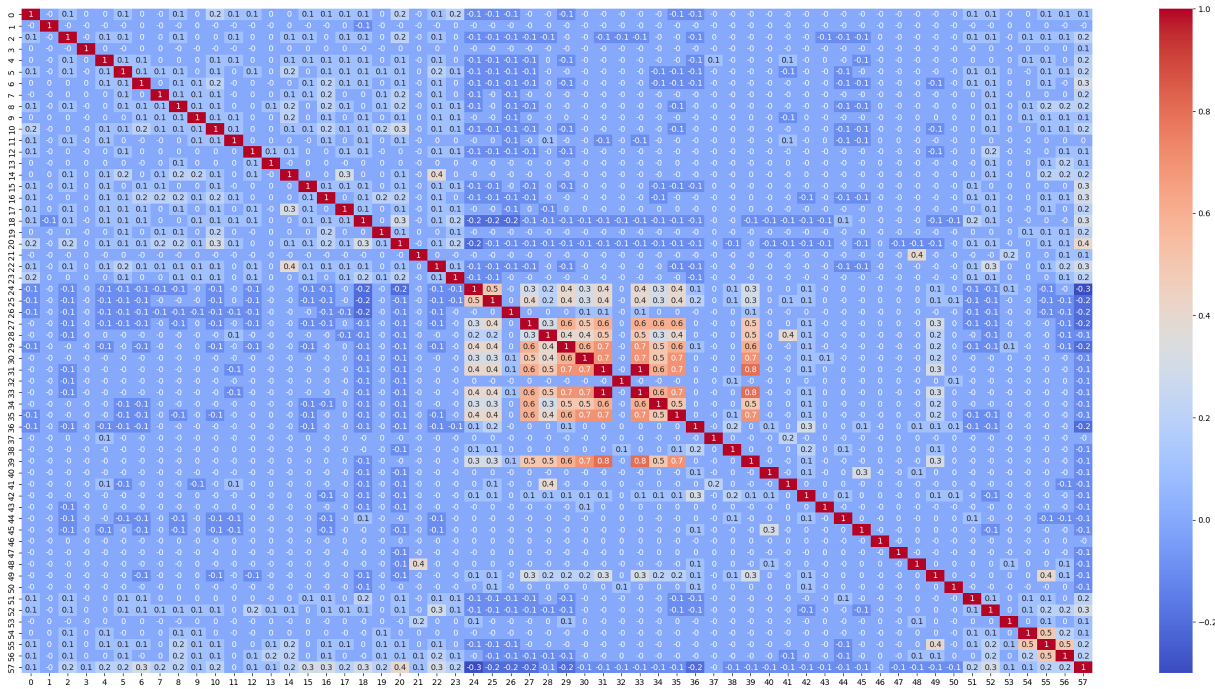
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	0.00	0.64	0.00	0.00	0.00	0.32
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	0.21	0.79	0.65	0.21	0.14	0.14
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	0.38	0.45	0.12	0.00	1.75	0.06

In [13]:
```
df.describe()
```

Out[13]:

	0	1	2	3	4	5	
count	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
std	0.305358	1.290575	0.504143	1.395151	0.672513	0.273824	0
min	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
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0
75%	0.000000	0.000000	0.420000	0.000000	0.380000	0.000000	0
max	4.540000	14.280000	5.100000	42.810000	10.000000	5.880000	7

In [14]:
```
plt.figure(figsize = (30, 15))
sns.heatmap(df.corr().round(1), annot=True, cmap='coolwarm')
plt.show()
```

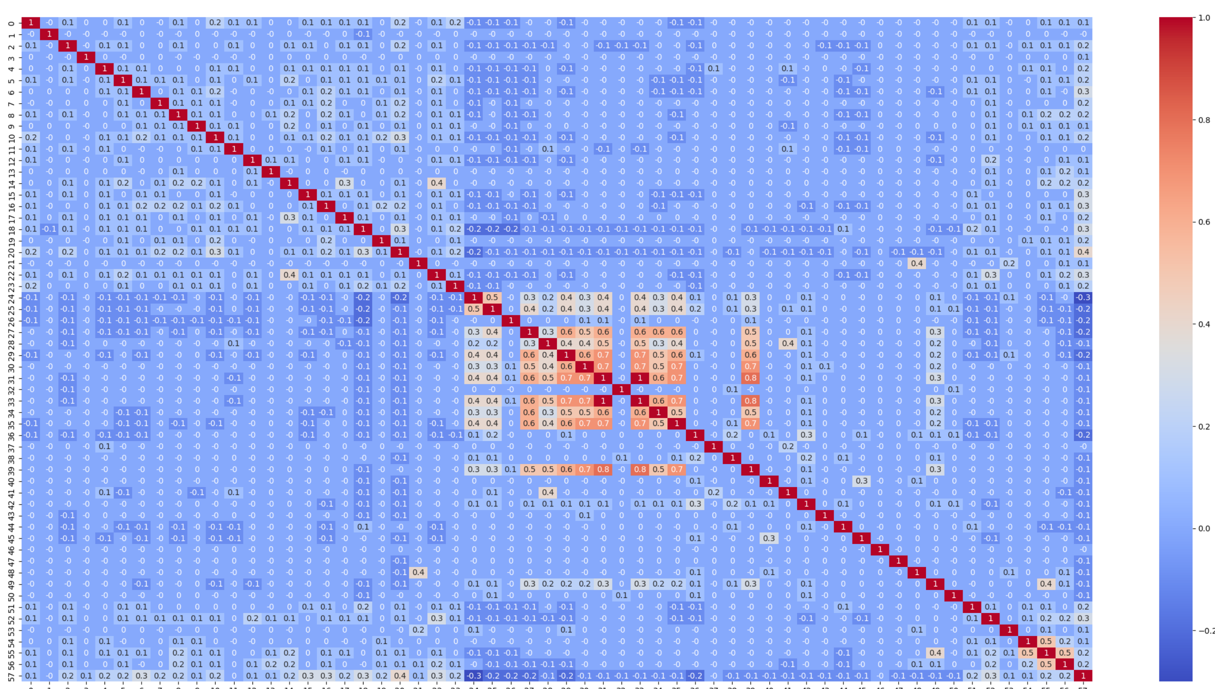


The heatmap shows feature #31 and #33 have 100% correlation; therefore, remove #31 and update #33 to be average of the two.

In [50]:
```
df2 = df.iloc[:, :-1]
df2[33] = (df2[31] + df2[33])/2
df2 = df2.drop(31, axis=1)
```

Now correlation matrix looks much better.

In [16]:
```
plt.figure(figsize = (30, 15))
sns.heatmap(df.corr().round(1), annot=True, cmap='coolwarm')
plt.show()
```



In [51]:
```
X = df2.values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

Decision Tree

First create a Decision Tree with default settings to observe.

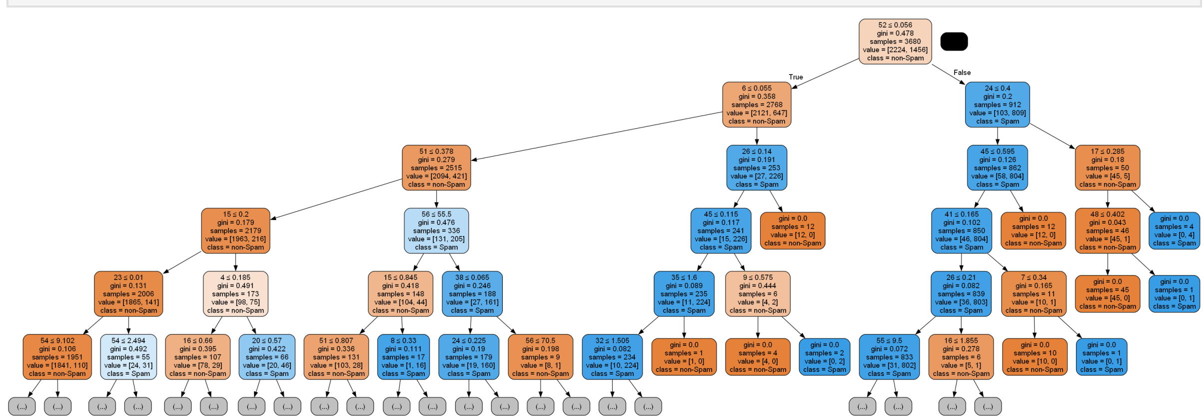
In [52]:
```
clf = DecisionTreeClassifier(random_state=1)
clf.fit(X_train, y_train)
y_pred_train = clf.predict(X_train)
y_pred_test = clf.predict(X_test)
```

In [55]:
```
features = df2.columns
classes = ["non-Spam", "Spam"]
```

In [82]:
```
# Confusion Matrix
def plot_confusionmatrix(y_pred, y, classes):
    cf = confusion_matrix(y_pred, y)
    sns.heatmap(cf, annot=True, yticklabels=classes, xticklabels=classes, cmap='coo
    plt.tight_layout()
    plt.show()

# Classification Report
def print_classification_metrics(y_pred, y):
    print("Classification Report:")
    print(classification_report(y, y_pred))
    error = 1 - accuracy_score(y, y_pred)
    print("Classification Error:", error)

# Decision Tree Graphics
def plot_decision_tree(model_name, out_file, max_depth, feature_names, class_names)
    treedot = tree.export_graphviz(model_name, out_file=out_file, max_depth=max_dep
                                   class_names=class_names, label='all', filled=Tru
                                   special_characters=True, precision=3)
    graph = pydotplus.graph_from_dot_data(treedot)
    return Image(graph.create_png())
```

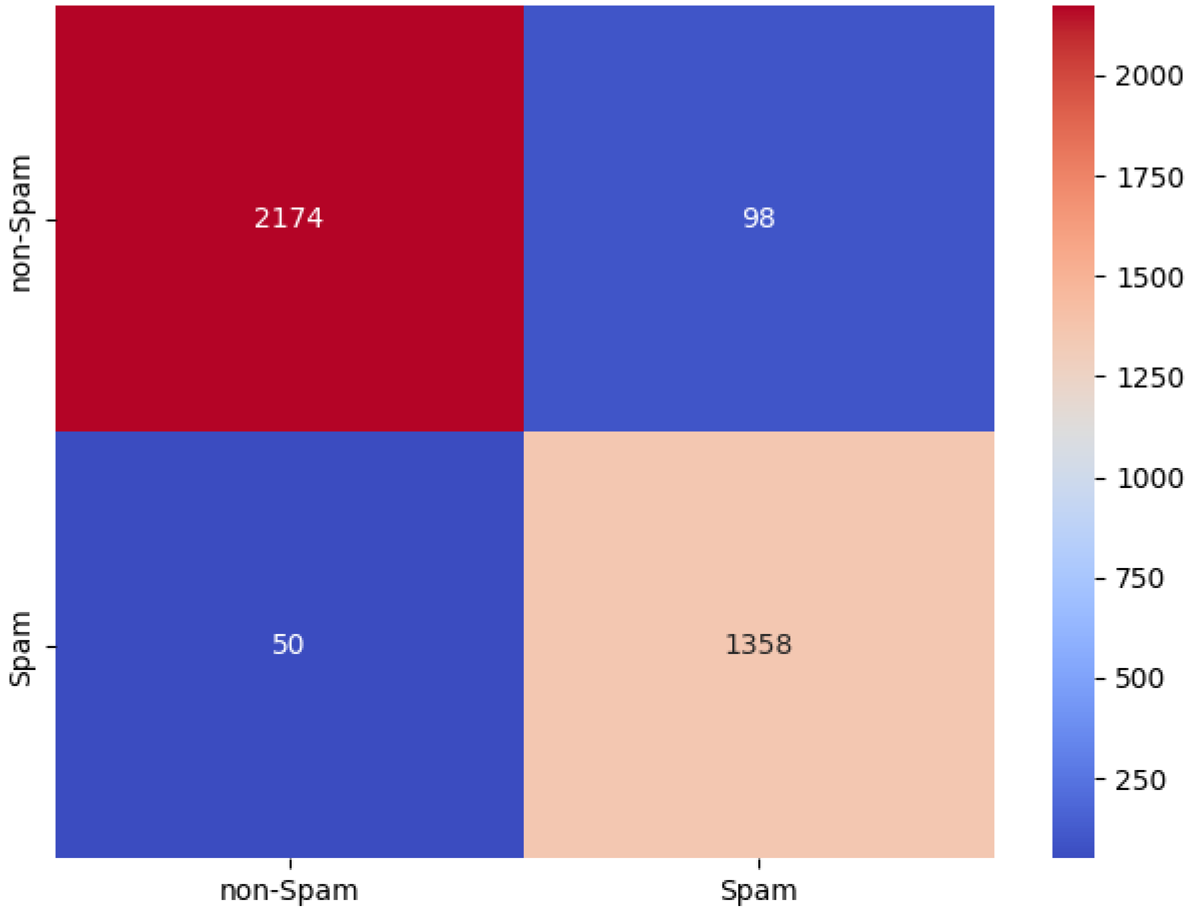
In [83]:
```
plot_decision_tree(clf, None, 5, features, classes)
```

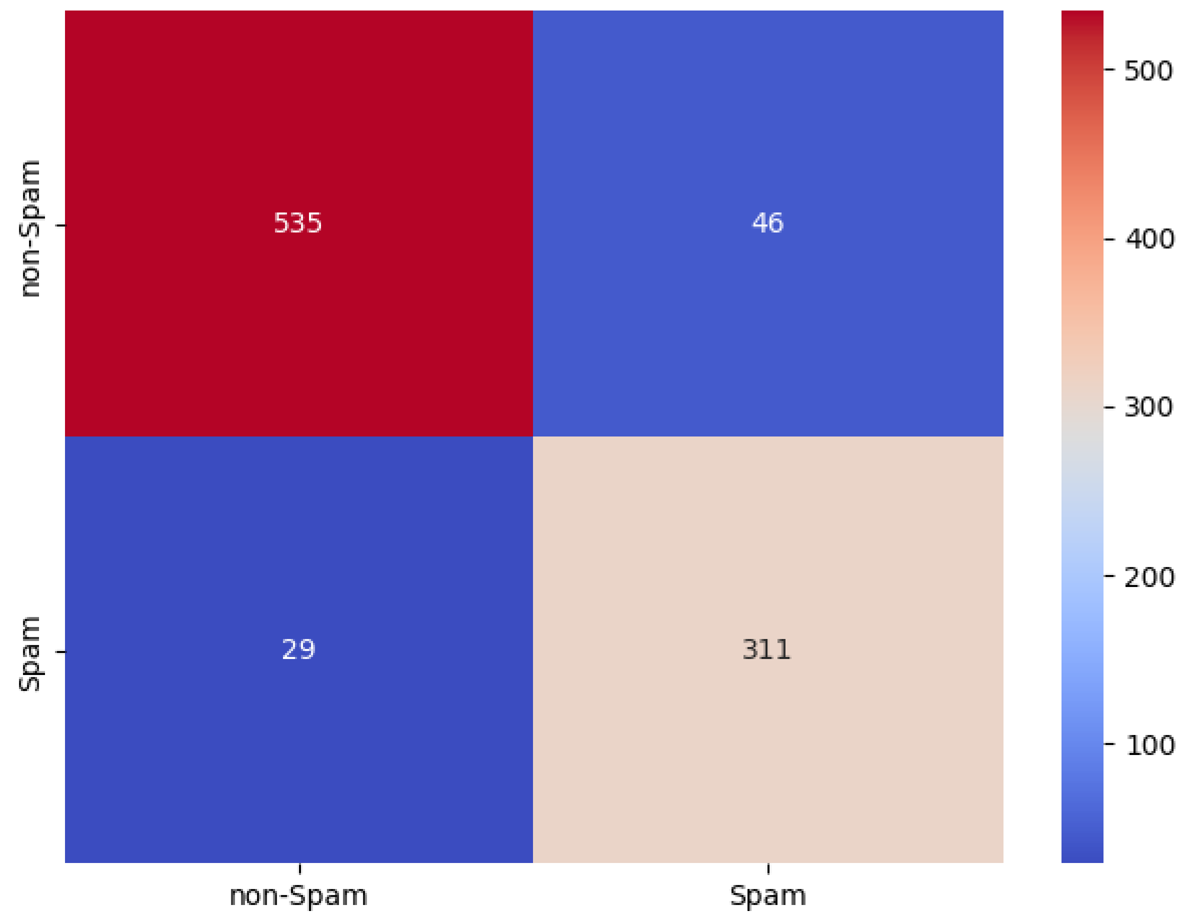
Out[83]:



In [77]:

```
plot_confusionmatrix(y_pred_train, y_train, classes=classes)
plot_confusionmatrix(y_pred_test, y_test, classes=classes)
print_classification_metrics(y_train, y_pred_train)
print_classification_metrics(y_test, y_pred_test)
```






```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      2272
           1       0.93      0.96      0.95      1408

    accuracy                           0.96      3680
   macro avg       0.96      0.96      0.96      3680
weighted avg       0.96      0.96      0.96      3680

Classification Error: 0.04021739130434787
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.92      0.93       581
           1       0.87      0.91      0.89       340

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

Classification Error: 0.08143322475570036
```

From Confusion Matrix and Classification Report, the model is overfitting - train data accuracy is significantly higher than test data; therefore, additional prunning is required.

Decision Tree - Pre Pruning

Pre-pruning uses GridSearchCV to find the best hyperparameters.

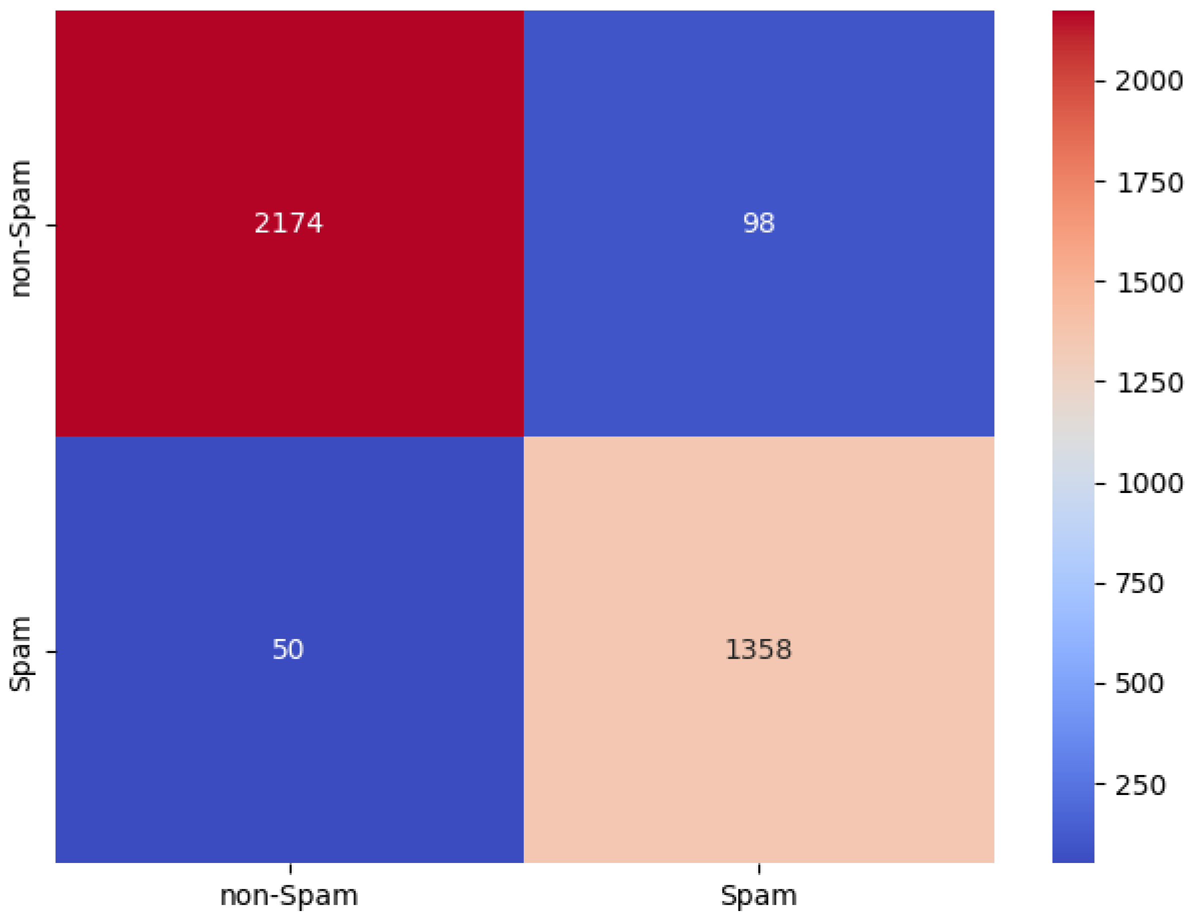
In [72]:
```
pipeline = Pipeline([
    ('clf', DecisionTreeClassifier(random_state=1))
])

parameters = [
    {'clf__max_depth': [4,5,6,7,8,9,10]},
    {'clf__min_samples_split': [2,3,4,5,6,7,8,9,10]},
    {'clf__min_samples_leaf': [2,3,4,5,6,7,8,9,10]}
]

cv_cart = GridSearchCV(pipeline,
                  param_grid=parameters,
                  scoring='accuracy',
                  cv=10, n_jobs=-1, verbose=2)

cv_cart.fit(X_train, y_train)
y_pred_cart = cv_cart.predict(X_test)

print(cv_cart.best_estimator_)
print(cv_cart.best_params_)
```

```
Fitting 10 folds for each of 25 candidates, totalling 250 fits
Pipeline(steps=[('clf', DecisionTreeClassifier(max_depth=9, random_state=1))])
{'clf__max_depth': 9}
```

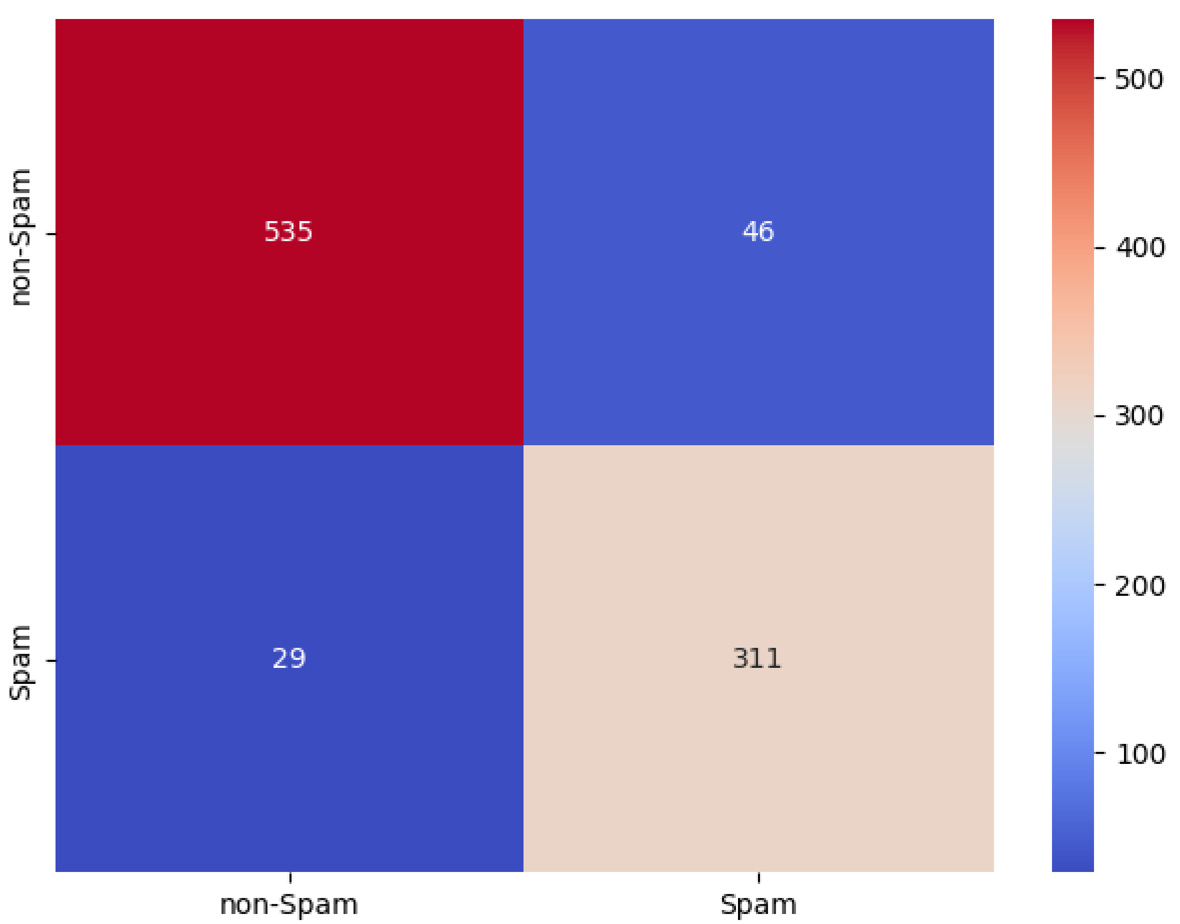
GridSearchCV determines max_depth=9 has highest accuracy and should be selected.

Recreate the model using this parameter and print out the metrics.

In [76]:
```
dt_model = DecisionTreeClassifier(max_depth=9, random_state=1)
dt_model.fit(X_train, y_train)
y_pred_train = dt_model.predict(X_train)
y_pred_test = dt_model.predict(X_test)
plot_confusionmatrix(y_pred_train, y_train, classes=classes)
plot_confusionmatrix(y_pred_test, y_test, classes=classes)
print_classification_metrics(y_train, y_pred_train)
print_classification_metrics(y_test, y_pred_test)
```



2174
98
50
1358
non-Spam
Spam
non-Spam
Spam
2000
1750
1500
1250
1000
750
500
250

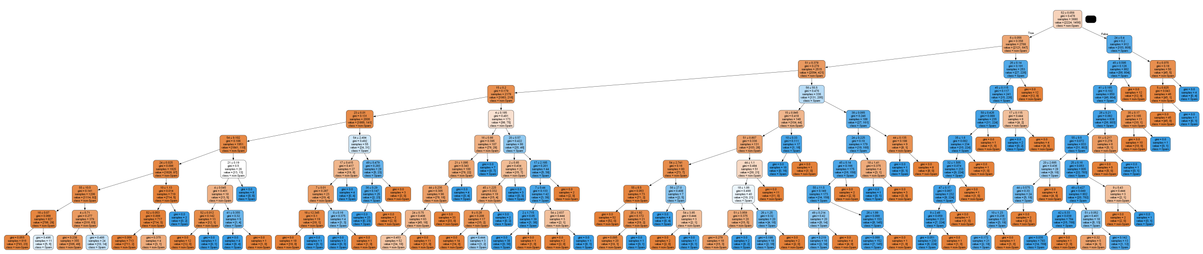


535
46
29
311
non-Spam
Spam
non-Spam
Spam
500
400
300
200
100

```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      2272
           1       0.93      0.96      0.95      1408

    accuracy                           0.96      3680
   macro avg       0.96      0.96      0.96      3680
weighted avg       0.96      0.96      0.96      3680

Classification Error: 0.04021739130434787
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.92      0.93       581
           1       0.87      0.91      0.89       340

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

Classification Error: 0.08143322475570036
```

In [84]:
```python
plot_decision_tree(dt_model, None, 9, features, classes)
```

Out[84]:



Post-pruning is needed to create a balanced tree.

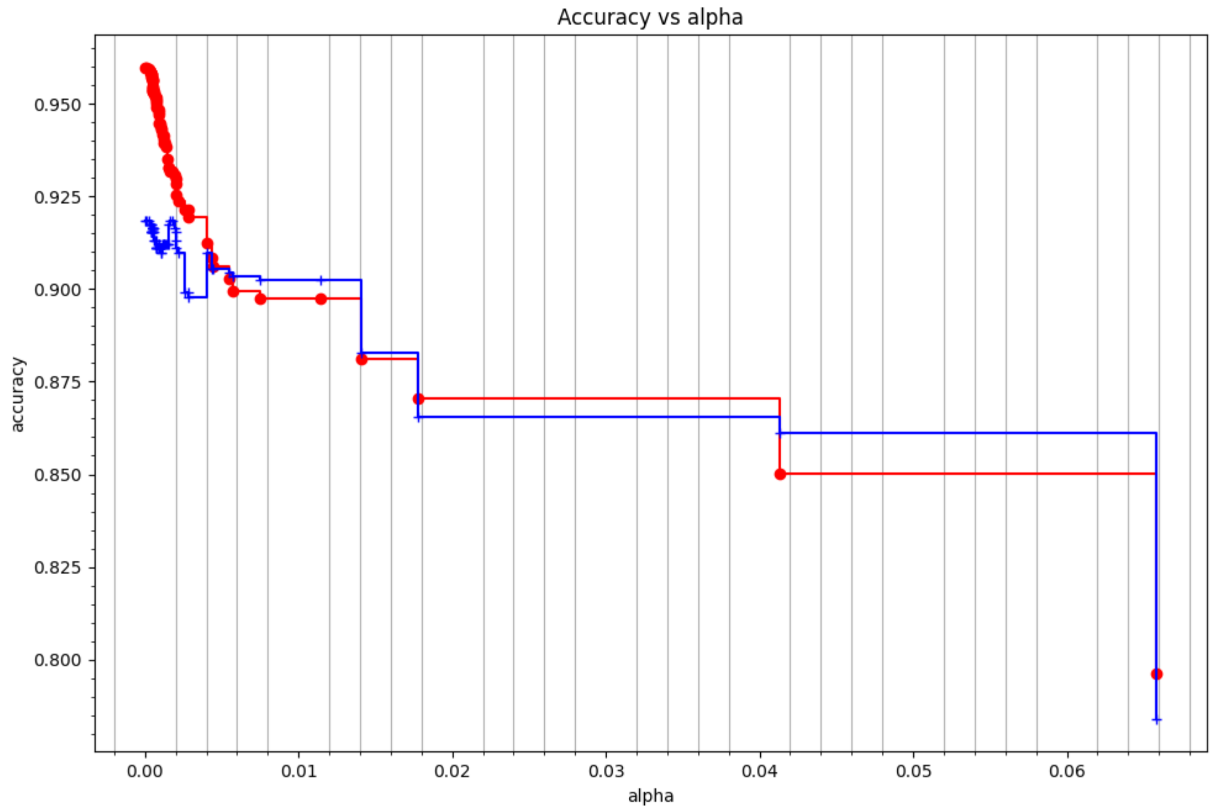
Decision Tree - Post-Pruning

In addition to hyper-parameters such as min_samples_leaf and max_depth to control the tree size, ccp_alpha parameter also limits tree growth: higher number will increase the number of nodes pruned.

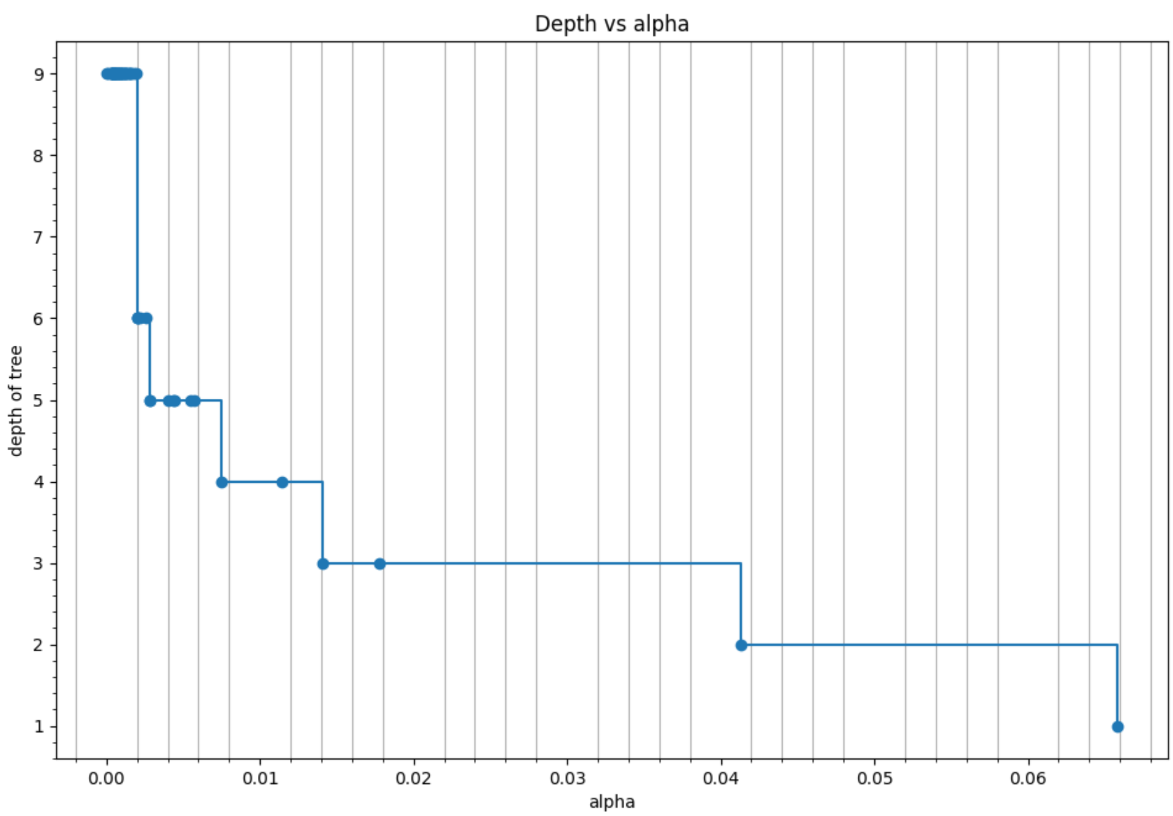
In [87]:
```python
path = dt_model.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities

# Maximum effective alpha value is removed, as it indicates only one node.
ccp_alphas, impurities = ccp_alphas[:-1], impurities[:-1]
```

In [88]:
```python
# Compare alpha impact on accuracy
clfs = []
for ccp_alpha in ccp_alphas:
    clf = DecisionTreeClassifier(max_depth=9, random_state=1, ccp_alpha=ccp_alpha)
    clf.fit(X_train, y_train)
    clfs.append(clf)
```

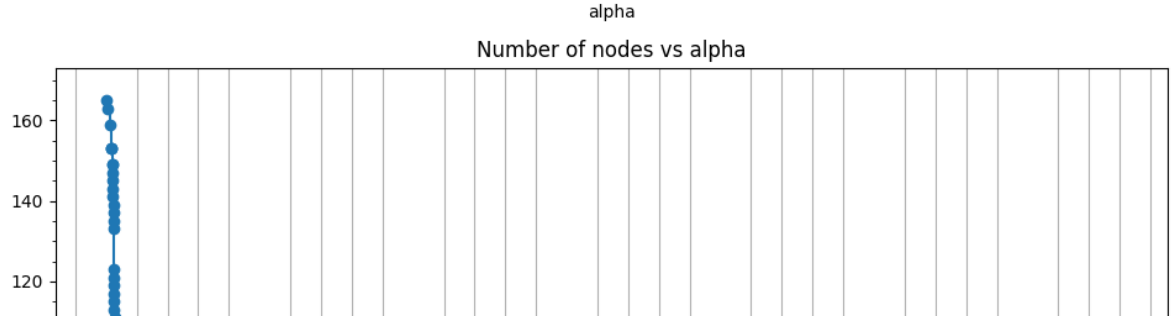
In [94]:
```
# Plot alpha and accuracy to look for a compromise
train_scores = [clf.score(X_train, y_train) for clf in clfs]
test_scores = [clf.score(X_test, y_test) for clf in clfs]

node_counts = [clf.tree_.node_count for clf in clfs]
depth = [clf.tree_.max_depth for clf in clfs]

fig, ax = plt.subplots(3, 1, figsize=(10,20))

ax[0].plot(ccp_alphas, train_scores, color="red", marker="o", label="train", drawst
ax[0].plot(ccp_alphas, test_scores, color="blue", marker="+", label="test", drawsty
ax[0].set_xlabel("alpha")
ax[0].set_ylabel("accuracy")
ax[0].set_title("Accuracy vs alpha")
ax[0].grid(axis = "x", which='minor')
ax[0].minorticks_on()

ax[1].plot(ccp_alphas, depth, marker="o", drawstyle="steps-post")
ax[1].set_xlabel("alpha")
ax[1].set_ylabel("depth of tree")
ax[1].set_title("Depth vs alpha")
ax[1].grid(axis = "x", which='minor')
ax[1].minorticks_on()

ax[2].plot(ccp_alphas, node_counts, marker="o", drawstyle="steps-post")
ax[2].set_xlabel("alpha")
ax[2].set_ylabel("number of nodes")
ax[2].set_title("Number of nodes vs alpha")
ax[2].grid(axis = "x", which='minor')
ax[2].minorticks_on()

fig.tight_layout()
```



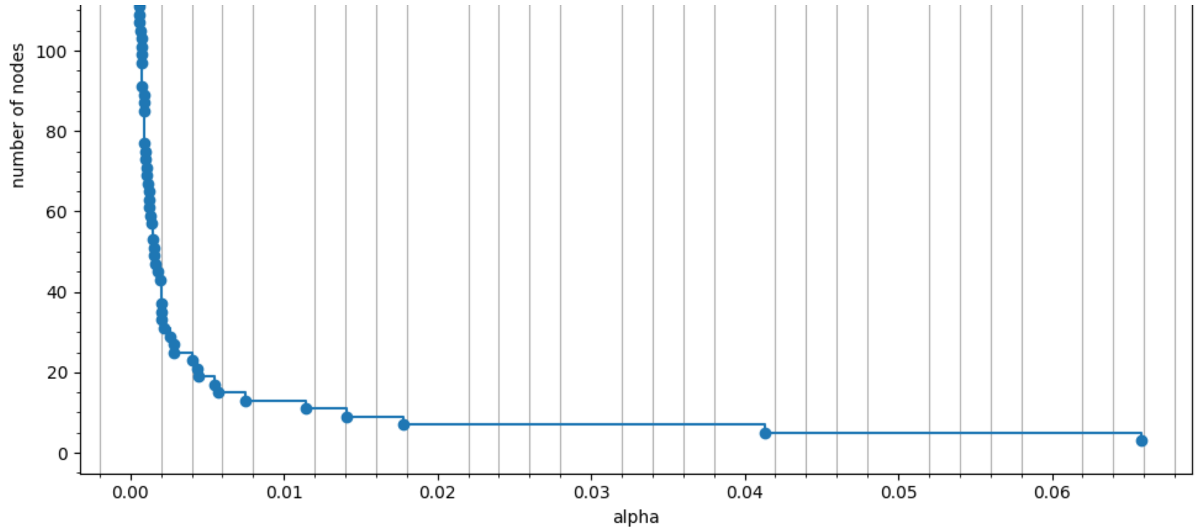
Accuracy vs alpha
accuracy
0.950
0.925
0.900
0.875
0.850
0.825
0.800
0.00
0.01
0.02
0.03
0.04
0.05
0.06
alpha



Depth vs alpha
depth of tree
9
8
7
6
5
4
3
2
1
0.00
0.01
0.02
0.03
0.04
0.05
0.06
alpha



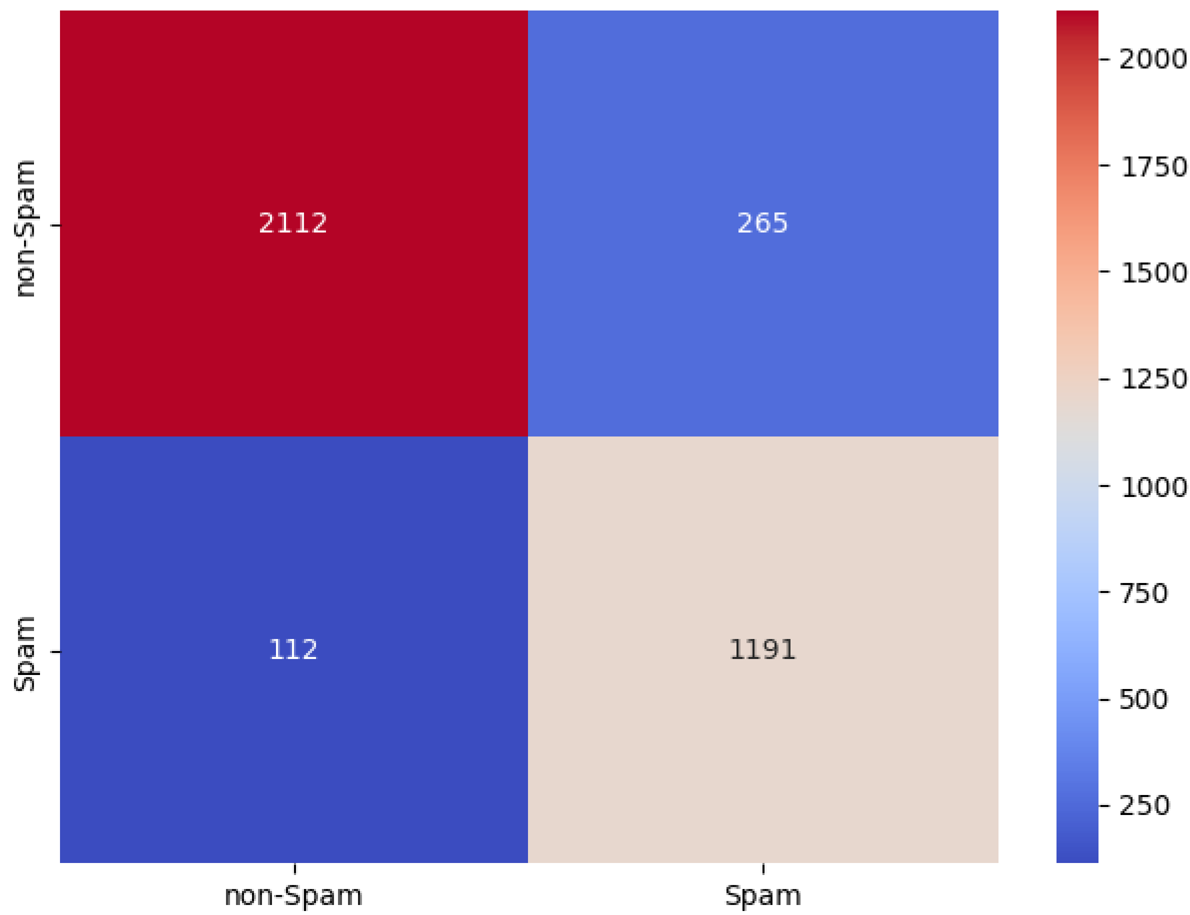
Number of nodes vs alpha
160
140
120

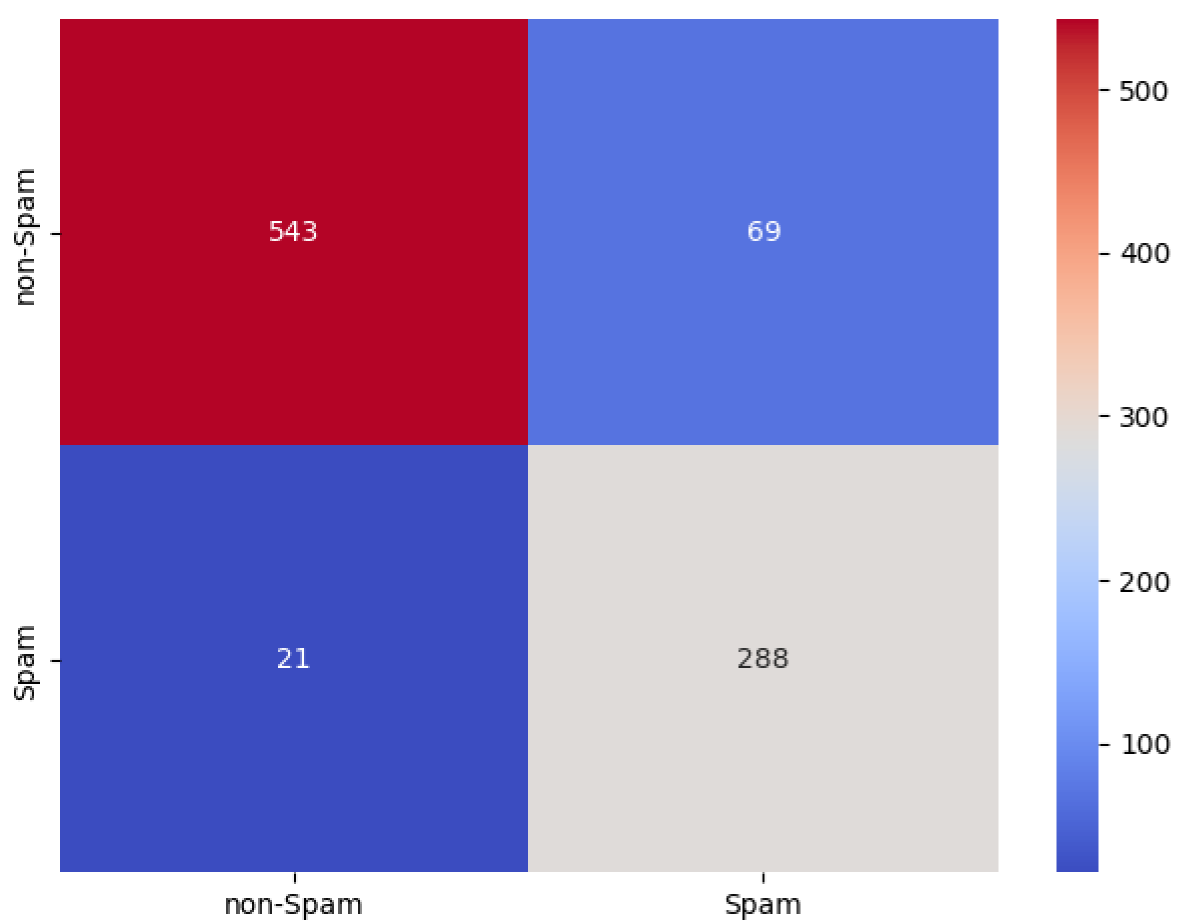


alpha is selected to be 0.014 when test data and train data accuracy difference is visiblly small and stable.

In [102…

```
dt_model = DecisionTreeClassifier(max_depth=9, random_state=1,ccp_alpha=0.014)
dt_model.fit(X_train, y_train)
y_pred_train = dt_model.predict(X_train)
y_pred_test = dt_model.predict(X_test)
plot_confusionmatrix(y_pred_train, y_train, classes=classes)
plot_confusionmatrix(y_pred_test, y_test, classes=classes)
print_classification_metrics(y_train, y_pred_train)
print_classification_metrics(y_test, y_pred_test)
```


non-Spam
Spam
2112
265
112
1191
non-Spam
Spam
2000
1750
1500
1250
1000
750
500
250

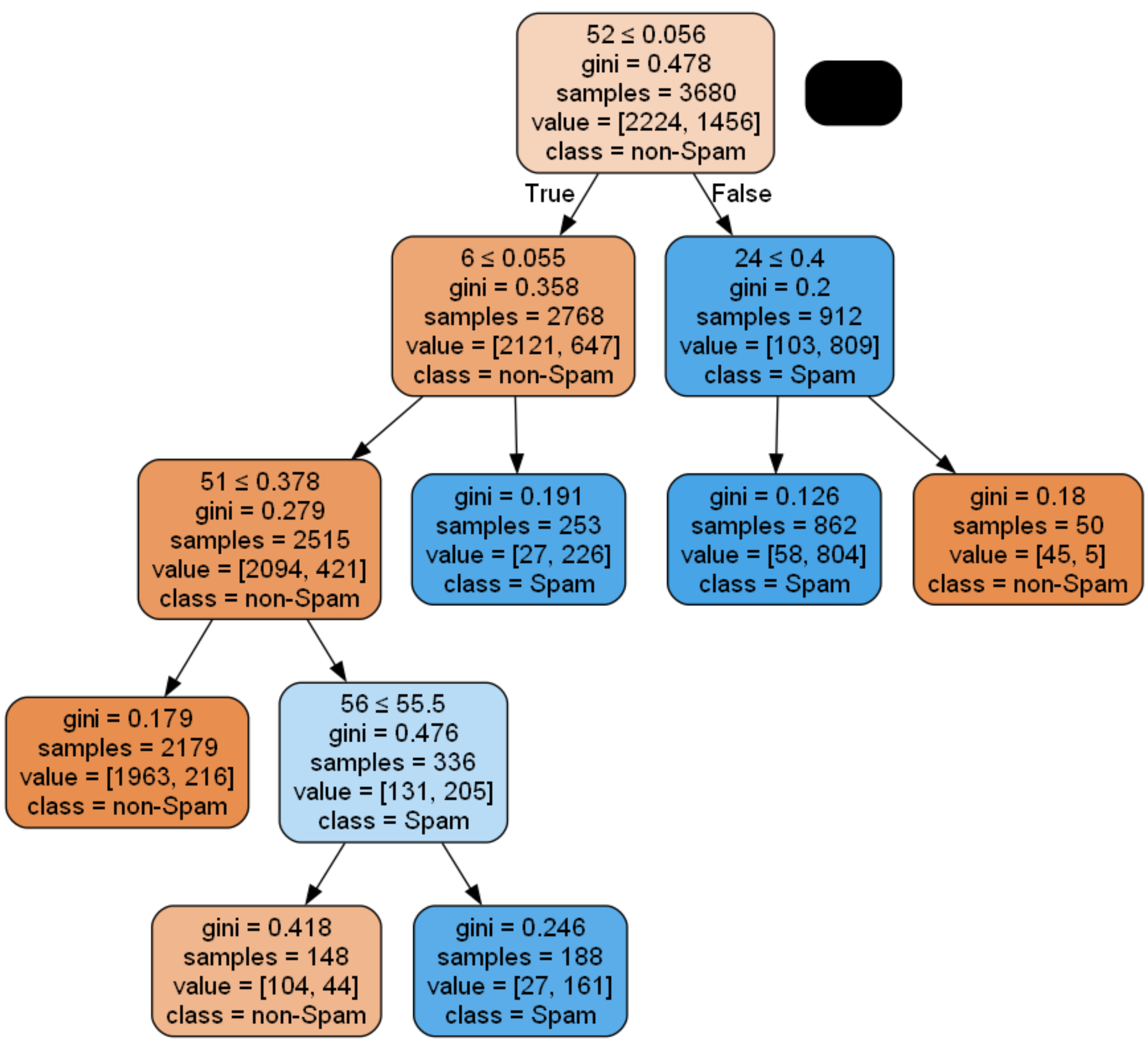

non-Spam
Spam
543
69
21
288
non-Spam
Spam
500
400
300
200
100

```
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.89      0.92      2377
           1       0.82      0.91      0.86      1303

    accuracy                           0.90      3680
   macro avg       0.88      0.90      0.89      3680
weighted avg       0.90      0.90      0.90      3680

Classification Error: 0.102445652173913
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.89      0.92       612
           1       0.81      0.93      0.86       309

    accuracy                           0.90       921
   macro avg       0.88      0.91      0.89       921
weighted avg       0.91      0.90      0.90       921

Classification Error: 0.09771986970684043
```

In [103…

```python
plot_decision_tree(dt_model, None, 9, features, classes)
```

Out[103]:



At last this Decision Tree looks reasonably well:

- Visually balanced and small tree which promote fast computation in production environment
- Compable accuracy for both train and test data which assure model performance stability

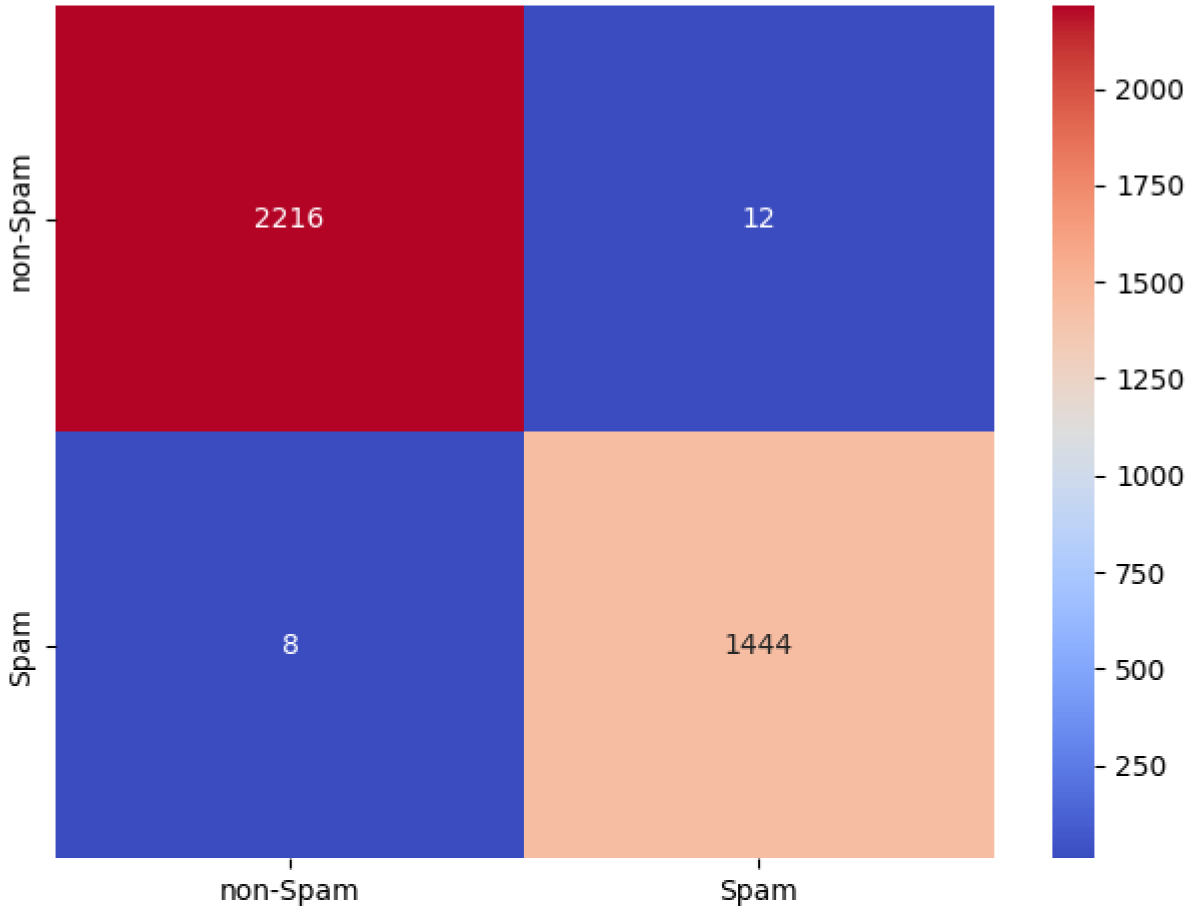
Random Forest

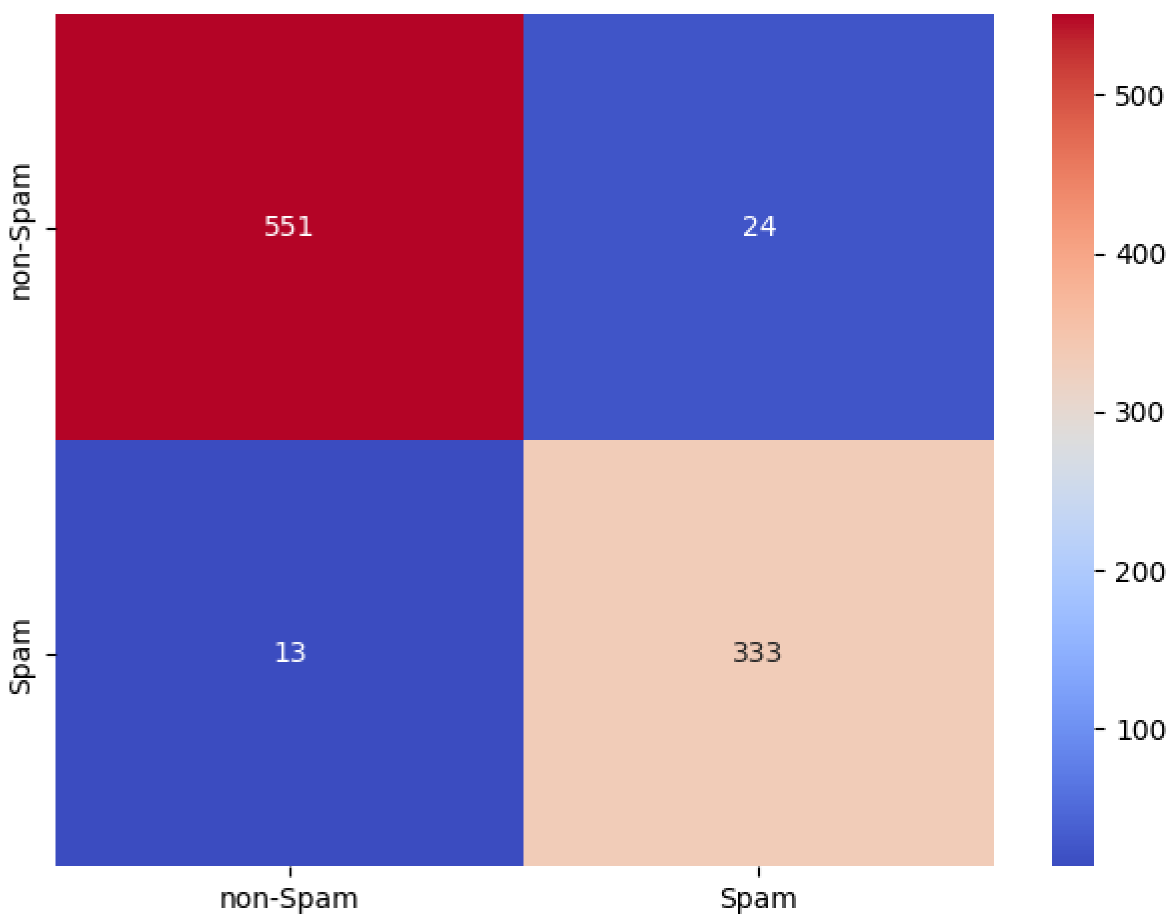
Random Forest doesn't require pruning as each tree is allowed to grow fully.

In [104…
```
pipeline = Pipeline([
    ('clf', RandomForestClassifier(random_state=1, n_jobs=-1, warm_start=True))
])

parameters = [
    {'clf__n_estimators': [2,5,10,20,30,40]},
    {'clf__max_depth': [3,4,5,6,7,8,9,10]},
    {'clf__min_samples_split': [2,3,4,5,6,7,8,9,10]},
    {'clf__min_samples_leaf': [2,3,4,5,6,7,8,9,10]}
]

cv_rf = GridSearchCV(pipeline,
                  param_grid=parameters,
                  scoring='accuracy',
                  cv=10, n_jobs=-1, verbose=2)

cv_rf.fit(X_train, y_train)
print(cv_rf.best_estimator_)
print(cv_rf.best_params_)
```

```
Fitting 10 folds for each of 32 candidates, totalling 320 fits
Pipeline(steps=[('clf',
                 RandomForestClassifier(min_samples_split=5, n_jobs=-1,
                                        random_state=1, warm_start=True))])
{'clf__min_samples_split': 5}
```

In [105…
```
y_pred_train_rf = cv_rf.predict(X_train)
y_pred_test_rf = cv_rf.predict(X_test)

plot_confusionmatrix(y_pred_train_rf, y_train, classes=classes)
plot_confusionmatrix(y_pred_test_rf, y_test, classes=classes)

print_classification_metrics(y_train, y_pred_train_rf)
print_classification_metrics(y_test, y_pred_test_rf)
```


non-Spam
Spam
2216
12
8
1444
non-Spam
Spam
2000
1750
1500
1250
1000
750
500
250


non-Spam
Spam
551
24
13
333
non-Spam
Spam
500
400
300
200
100

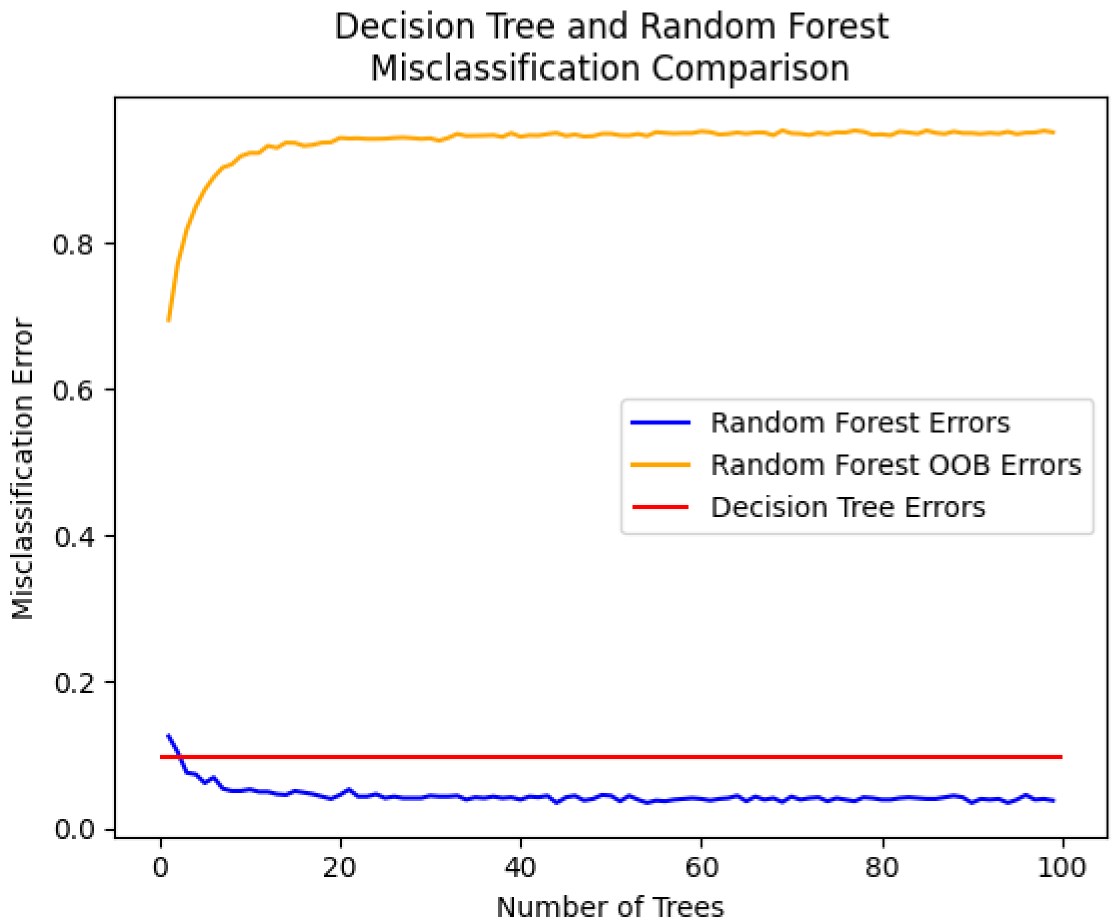
```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      2228
           1       0.99      0.99      0.99      1452

    accuracy                           0.99      3680
   macro avg       0.99      0.99      0.99      3680
weighted avg       0.99      0.99      0.99      3680

Classification Error: 0.005434782608695676
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       575
           1       0.93      0.96      0.95       346

    accuracy                           0.96       921
   macro avg       0.95      0.96      0.96       921
weighted avg       0.96      0.96      0.96       921

Classification Error: 0.04017372421281218
```

Computationally Random Forest is much simpler to implement and the result is impressive - the first attempt has come to comparable accuracy rates for both train and test datasets.

Misclassification Rate Comparison

In [117…

```
list_tree = []
list_error = []
list_oob = []

for num_tree in range(1,100,1):
    clf_rf = RandomForestClassifier(n_estimators=num_tree, n_jobs=-1, warm_start=Tr
    clf_rf.fit(X_train, y_train)
    y_pred = clf_rf.predict(X_test)
    error = 1 - accuracy_score(y_test, y_pred)
    list_tree.append(num_tree)
    list_error.append(error)
    list_oob.append(clf_rf.oob_score_)

plt.plot(list_tree, list_error, color="blue", label='Random Forest Errors')
plt.plot(list_tree, list_oob, color="orange", label='Random Forest OOB Errors')
plt.hlines(y=0.09771986970684043,xmin=0, xmax=100,color="red", label="Decision Tree
plt.xlabel("Number of Trees")
plt.ylabel("Misclassification Error")
plt.title("Decision Tree and Random Forest\nMisclassification Comparison")
plt.legend()
plt.show()
```

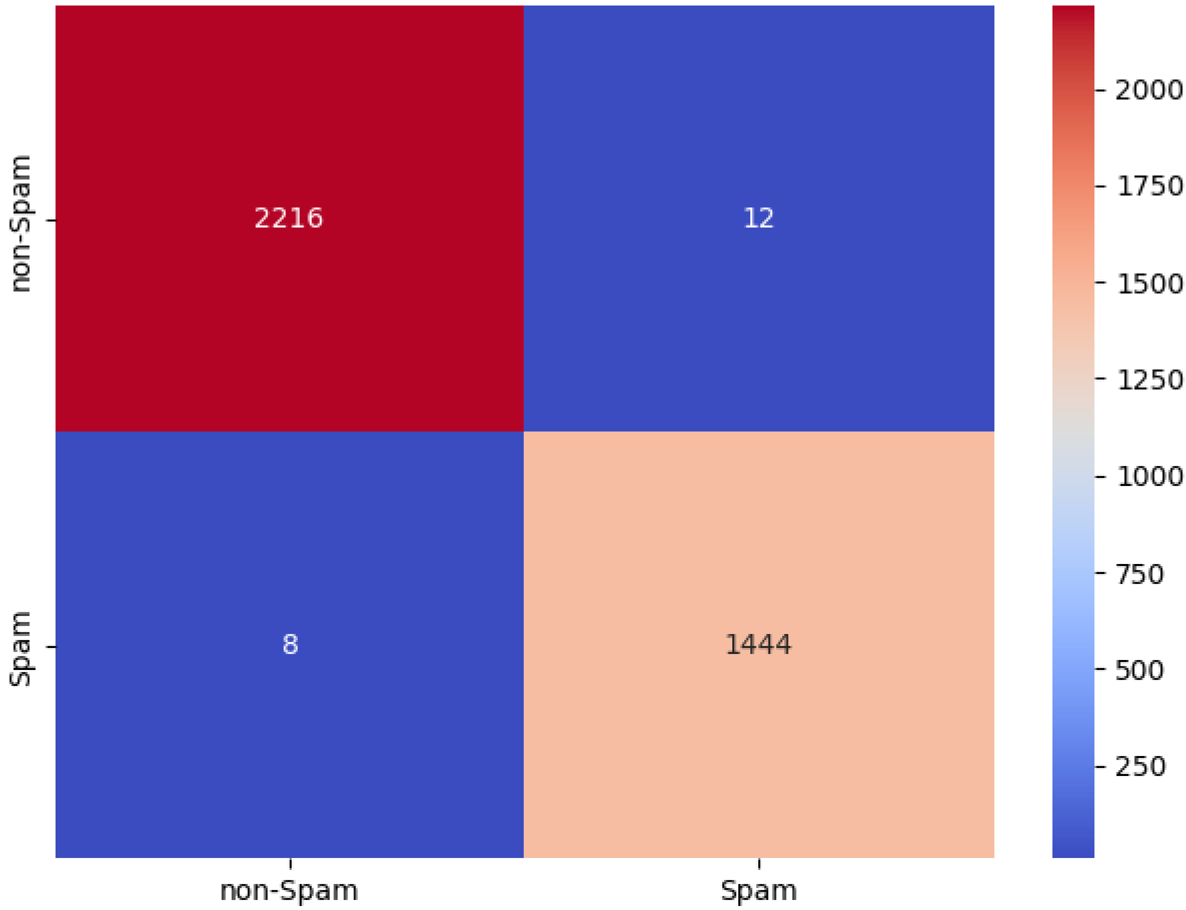


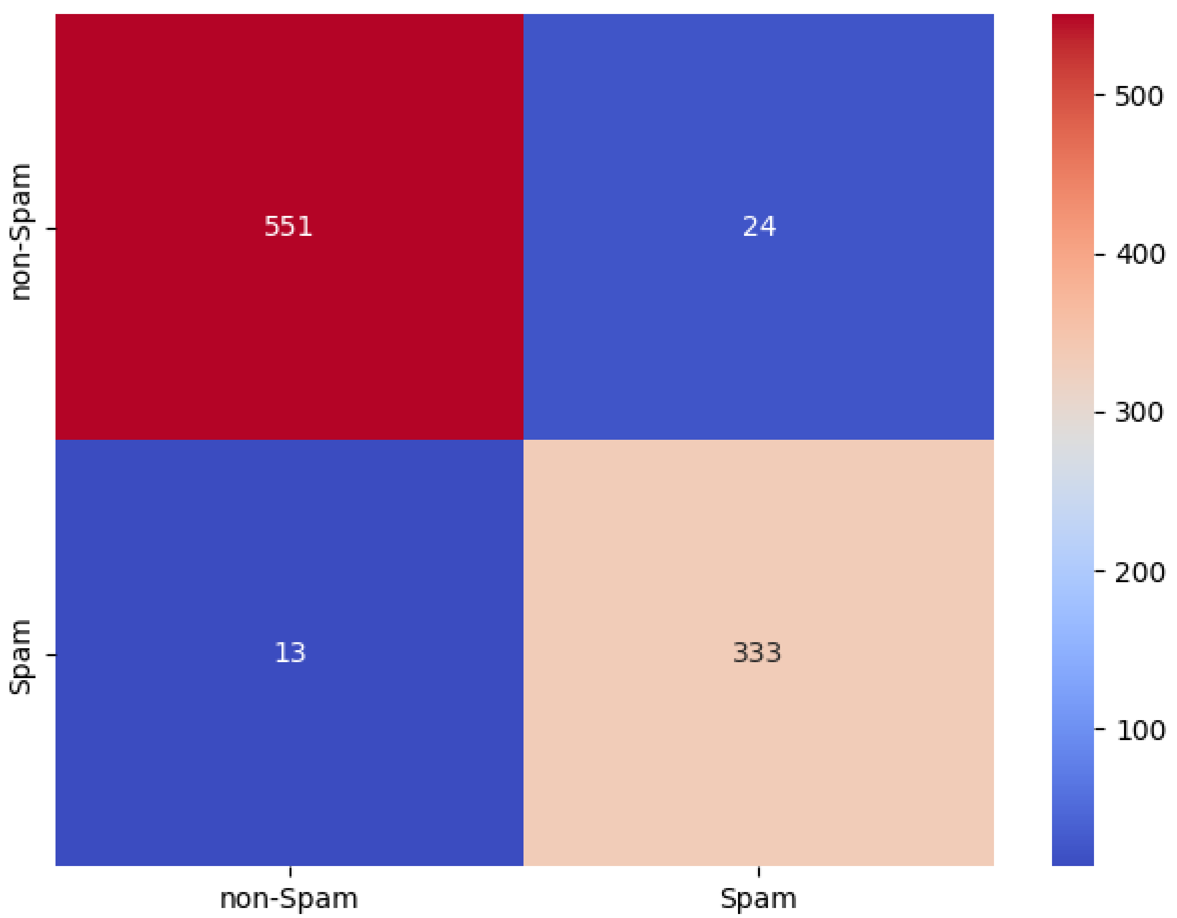
It is clear Random Forest performs significantly better than traditional Decision Tree. OOB error for Random Forest also indicates it is effective for unseen data with 20+ trees. However, it has the shortcoming of being difficult to explain, while Decision Tree is well understood and easy to explain.

Finally create Random Forest with the new parameter.

In [118…

```
clf_rf = RandomForestClassifier(n_estimators=20, n_jobs=-1, warm_start=True, oob_sc
clf_rf.fit(X_train, y_train)
y_pred_train = clf_rf.predict(X_train)
y_pred_test = clf_rf.predict(X_test)
plot_confusionmatrix(y_pred_train_rf, y_train, classes=classes)
plot_confusionmatrix(y_pred_test_rf, y_test, classes=classes)
print_classification_metrics(y_train, y_pred_train_rf)
print_classification_metrics(y_test, y_pred_test_rf)
```


non-Spam
Spam
2216
12
8
1444
non-Spam
Spam
2000
1750
1500
1250
1000
750
500
250


non-Spam
Spam
551
24
13
333
non-Spam
Spam
500
400
300
200
100

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00      2228
           1       0.99      0.99      0.99      1452

    accuracy                           0.99      3680
   macro avg       0.99      0.99      0.99      3680
weighted avg       0.99      0.99      0.99      3680

Classification Error: 0.005434782608695676
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       575
           1       0.93      0.96      0.95       346

    accuracy                           0.96       921
   macro avg       0.95      0.96      0.96       921
weighted avg       0.96      0.96      0.96       921

Classification Error: 0.04017372421281218
```