In [ ]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel
from sklearn.inspection import permutation_importance
from sklearn.model_selection import train_test_split
```

In [ ]:
```python
df = pd.read_csv("dataset_full.csv")
data = np.loadtxt("dataset_full.csv", delimiter = ",", skiprows=1)
print(data.shape)
```

```
(88647, 112)
```

1. Preprocessing

In [ ]:
```python
supervised_y_values = data[:, -1]
x = data[:,0:-1]
feature_names = df.columns
print("Shape of feature_names is ", feature_names.shape)
print("Shape of x is ", x.shape)
print("Shape of supervised_y_values is ", supervised_y_values.shape)

# Splitting the data into a training and testing dataset

train_percent = 0.8
""""
test_percent = 1-train_percent
x_train = x[0:int(train_percent*len(x))]
y_train = supervised_y_values[0:int(train_percent*len(supervised_y_values))]
x_test = x[int(-test_percent*len(x)): :]
y_test = supervised_y_values[int(-test_percent*len(supervised_y_values)): :]
print("Shape of x_train is ", np.shape(x_train))
print("Shape of y_train is ", np.shape(y_train))
"""

x_train, x_test, y_train, y_test = train_test_split(x, supervised_y_values, train_size
```

```
Shape of feature_names is  (112,)
Shape of x is  (88647, 111)
Shape of supervised_y_values is  (88647,)
```

Tree Based Feature Selection

Below, tree-based feature selection is used as a pre-processing step. Essentially, all the data points are placed at the root of a tree. The tree expands by splitting the data among different features. The first split in the data (going from root node to subsequent nodes) occurs among the feature which decreases the gini impurity the most; decreasing impurity just means trying to get the data at each node to be as close to homogenous as possible. Subsequent splits decrease the impurity further until ideally, every leaf is either fully "spam" or fully "not spam". Multiple trees are creataed in this fashion, and the average accuracy (calculated from gini coefficient) is assigned to each feature. The following link explains decreasing impurity well: https://www.baeldung.com/cs/impurity-entropy-gini-index

The min_samples_leaf parameter allows us to adjust the minimum number of data points in a leaf. By increasing this parameter, the leaves may not be fully homogenous. This will decrease the training accuracy, but it will likely decrease overfitting and increase the test accuracy. After computing the impurity-based feature importances, the SelectFromModel Transformer discards the irrelevant features.

The above process is based on the detail provided in section 1.13.4.2 at the following link: https://scikit-learn.org/stable/modules/feature_selection.html#feature-selection

In [ ]:
```
forest = RandomForestClassifier(random_state=0, max_features=None)
# Setting max_features=None makes the tree consider all the features when looking for
# Currently max_samples=None, which means all the data points are considered. If this
forest.fit(x_train, y_train)

print("Training accuracy is ", forest.score(x_train, y_train))
print("Testing accuracy is ", forest.score(x_test, y_test))
```

```
Training accuracy is  0.9999435960348012
Testing accuracy is  0.968020304568528
```

In [ ]:
```
# Now, we'll determine the most important features
importances = forest.feature_importances_
important_features = np.argsort(importances)
numDesiredFeatures = 2

# Now, extracting only the important 2 features and points from x_train
simplified_model_2 = SelectFromModel(forest, prefit=True, max_features=numDesiredFeatu
simplified_x_train_2 = simplified_model_2.transform(x_train)
print("Results for 2 most important features")
print("Shape of simplified_x_train_2 is ", np.shape(simplified_x_train_2))
print("The importances of the most important ", numDesiredFeatures, " features are ",
sum_2_important_features_forest = np.sum(importances[important_features[-numDesiredFea
print("These features are ", feature_names[important_features[-numDesiredFeatures: :]]
print("The total importance is ", np.sum(importances))
```

```
Results for 2 most important features
Shape of simplified_x_train_2 is  (70917, 2)
The importances of the most important  2  features are  [0.10994991 0.5977344 ]
These features are  Index(['time_domain_activation', 'directory_length'], dtype='obje
ct')
The total importance is  0.9999999999999997
```

In [ ]:
```
# Now, we'll determine the most important features
importances = forest.feature_importances_
important_features = np.argsort(importances)
numDesiredFeatures = 3

# Now, extracting only the important 3 features and points from x_train
simplified_model_3 = SelectFromModel(forest, prefit=True, max_features=numDesiredFeatu
simplified_x_train_3 = simplified_model_3.transform(x_train)
print("Results for 3 most important features")
print("Shape of simplified_x_train_3 is ", np.shape(simplified_x_train_3))
print("The importances of the most important ", numDesiredFeatures, " features are ",
sum_3_important_features_forest = np.sum(importances[important_features[-numDesiredFea
```

```
print("These features are ", feature_names[important_features[-numDesiredFeatures: :]
print("The total importance is ", np.sum(importances))
```

```
Results for 3 most important features
Shape of simplified_x_train_3 is  (70917, 3)
The importances of the most important  3  features are  [0.03405149 0.10994991 0.5977
344 ]
These features are  Index(['asn_ip', 'time_domain_activation', 'directory_length'], d
type='object')
The total importance is  0.9999999999999997
```

In [ ]:

```
# Now, extracting the most important 4 features and points from x_train
numDesiredFeatures = 4
simplified_model_4 = SelectFromModel(forest, prefit=True, max_features=numDesiredFeatu
simplified_x_train_4 = simplified_model_4.transform(x_train)
print("Results for 4 most important features")
print("Shape of simplified_x_train_4 is ", np.shape(simplified_x_train_4))
print("The importances of the most important ", numDesiredFeatures, " features are ",
sum_4_important_features_forest = np.sum(importances[important_features[-numDesiredFea
print("These features are ", feature_names[important_features[-numDesiredFeatures: :]
print("The total importance is ", np.sum(importances))
```

```
Results for 4 most important features
Shape of simplified_x_train_4 is  (70917, 4)
The importances of the most important  4  features are  [0.02911747 0.03405149 0.1099
4991 0.5977344 ]
These features are  Index(['time_response', 'asn_ip', 'time_domain_activation',
       'directory_length'],
      dtype='object')
The total importance is  0.9999999999999997
```

In [ ]:

```
# Now, extracting the most important 5 features and points from x_train
numDesiredFeatures = 5
simplified_model_5 = SelectFromModel(forest, prefit=True, max_features=numDesiredFeatu
simplified_x_train_5 = simplified_model_5.transform(x_train)
print("Results for 5 most important features")
print("Shape of simplified_x_train_5 is ", np.shape(simplified_x_train_5))
print("The importances of the most important ", numDesiredFeatures, " features are ",
sum_5_important_features_forest = np.sum(importances[important_features[-numDesiredFea
print("These features are ", feature_names[important_features[-numDesiredFeatures: :]
print("The total importance is ", np.sum(importances))
```

```
Results for 5 most important features
Shape of simplified_x_train_5 is  (70917, 5)
The importances of the most important  5  features are  [0.02610967 0.02911747 0.0340
5149 0.10994991 0.5977344 ]
These features are  Index(['length_url', 'time_response', 'asn_ip', 'time_domain_acti
vation',
       'directory_length'],
      dtype='object')
The total importance is  0.9999999999999997
```

Feature Permutations

The previous method of tree based selection is slightly biased towards high-cardinality features (features which may have many unique values). There is more information present for features with several different values, thereby ranking those features as more important than ones with less cardinality. Our dataset may be susceptible to this, as the number of certain characters within a url may vary tremendously between websites.

Feature permutations is not as biased towards high-cardinality features. In feature permutations, a model score (below, accuracy, precision, and recall are the external measures) with all the features is calculated. Then, a feature is removed, and the new model score is calculated. The difference between the two values is the importance of that feature to the model. This process is repeated for all the features. Through this procedure, the importance of a feature avoids high-cardinality bias since it reflects its importance to the model as a whole, not its intrinsic predictive value.

Therefore, the below cell ranks feature importance based on feature permutations. This computation is much more costly, and more information about it can be found at the following link: https://scikit-learn.org/stable/modules/permutation_importance.html#permutation-importance

THERE ARE 3 CELLS BELOW THIS CELL, ALL OF WHICH BUILD A FEATURE PERMUTATION MODEL. PLEASE ONLY RUN ONE OF THEM. EACH ONE IS DIFFERENT. 1) The first one is the original one we used 2) This one is like the first one, but I pass in the test data instead. This is just to see whether the "important" features are just overfitted features. 3) This one is like the first one, but the min_samples_leaf parameter in the RandomForestClassifier allows for more points at each leaf node. This reduces overfitting.

In [ ]:
```
# Just FYI, this cell took about 11 minutes to run on my local machine

perm_model = permutation_importance(forest, x_train, y_train, n_repeats=10, random_sta
perm_model_accuracy = perm_model["accuracy"]
perm_model_precision = perm_model["precision"]
perm_model_recall = perm_model["recall"]
print(type(perm_model_accuracy))
perm_importances_1_accuracy = perm_model_accuracy.importances_mean
perm_importances_1_precision = perm_model_precision.importances_mean
perm_importances_1_recall = perm_model_recall.importances_mean
perm_important_features_1_accuracy = np.argsort(perm_importances_1_accuracy)
perm_important_features_1_precision = np.argsort(perm_importances_1_precision)
perm_important_features_1_recall = np.argsort(perm_importances_1_recall)
```

```
<class 'sklearn.utils._bunch.Bunch'>
```

In [ ]:
```
# Just FYI, this cell took about 3 minutes to run on my local machine
# Doing the same operation as above, but inserting the test data instead

perm_model = permutation_importance(forest, x_test, y_test, n_repeats=10, random_state
perm_model_accuracy = perm_model["accuracy"]
perm_model_precision = perm_model["precision"]
perm_model_recall = perm_model["recall"]
print(type(perm_model_accuracy))
perm_importances_2_accuracy = perm_model_accuracy.importances_mean
perm_importances_2_precision = perm_model_precision.importances_mean
perm_importances_2_recall = perm_model_recall.importances_mean
perm_important_features_2_accuracy = np.argsort(perm_importances_2_accuracy)
perm_important_features_2_precision = np.argsort(perm_importances_2_precision)
perm_important_features_2_recall = np.argsort(perm_importances_2_recall)
```

```
<class 'sklearn.utils._bunch.Bunch'>
```

In [ ]:
```
# Just FYI, this cell took about 11 minutes to run on my local machine
# Increasing the min_samples_leaf parameter in RandomForestClassifier to decrease over

forest = RandomForestClassifier(random_state=0, max_features=None, min_samples_leaf=5)
# Setting max_features=None makes the tree consider all the features when looking for
# Currently max_samples=None, which means all the data points are considered. If this
forest.fit(x_train, y_train)
print("Training accuracy is ", forest.score(x_train, y_train))
print("Testing accuracy is ", forest.score(x_test, y_test))

perm_model = permutation_importance(forest, x_train, y_train, n_repeats=10, random_sta
perm_model_accuracy = perm_model["accuracy"]
perm_model_precision = perm_model["precision"]
perm_model_recall = perm_model["recall"]
print(type(perm_model_accuracy))
perm_importances_3_accuracy = perm_model_accuracy.importances_mean
perm_importances_3_precision = perm_model_precision.importances_mean
perm_importances_3_recall = perm_model_recall.importances_mean
perm_important_features_3_accuracy = np.argsort(perm_importances_3_accuracy)
perm_important_features_3_precision = np.argsort(perm_importances_3_precision)
perm_important_features_3_recall = np.argsort(perm_importances_3_recall)
```

```
Training accuracy is  0.9834031332402667
Testing accuracy is  0.9653130287648054
<class 'sklearn.utils._bunch.Bunch'>
```

In [ ]:
```
# Now, extracting the most important 2 features and points from x_train on normal Rand
numDesiredFeatures = 2
simplified_perm_x_train_accuracy_2 = x_train[:, perm_important_features_1_accuracy[-nu
simplified_perm_x_train_precision_2 = x_train[:, perm_important_features_1_precision[-
simplified_perm_x_train_recall_2 = x_train[:, perm_important_features_1_recall[-numDes
print("Results for 2 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_1_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_1_accuracy))
sum_2_important_features_perm_accuracy = np.sum(perm_importances_1_accuracy[perm_impor
print("The fraction of importance covered by the 2 features is ", sum_2_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_1_precision[-numDes
print("The total importance is ", np.sum(perm_importances_1_precision))
sum_2_important_features_perm_precision = np.sum(perm_importances_1_precision[perm_imp
print("The fraction of importance covered by the 2 features is ", sum_2_important_feat


print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_1_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_1_recall))
sum_2_important_features_perm_recall = np.sum(perm_importances_1_recall[perm_important
print("The fraction of importance covered by the 2 features is ", sum_2_important_feat
```

```
Results for 2 most important features
The importances of the most important  2  features for accuracy are  [0.09151966 0.19
200756]
These features are  Index(['time_domain_activation', 'directory_length'], dtype='obje
ct')
The total importance is  0.4340059506183266
The fraction of importance covered by the 2 features is  0.6532795726873419
The importances of the most important  2  features for precision are  [0.09374938 0.2
9131681]
These features are  Index(['time_domain_activation', 'directory_length'], dtype='obje
ct')
The total importance is  0.6342580139116339
The fraction of importance covered by the 2 features is  0.6071128446036258
The importances of the most important  2  features for recall are  [0.17970896 0.2444
9109]
These features are  Index(['time_domain_activation', 'directory_length'], dtype='obje
ct')
The total importance is  0.6075857008926758
The fraction of importance covered by the 2 features is  0.6981732065826276
```

In [ ]:

```
# Now, extracting the most important 3 features and points from x_train on normal Rand
numDesiredFeatures = 3
simplified_perm_x_train_accuracy_3 = x_train[:, perm_important_features_1_accuracy[-nu
simplified_perm_x_train_precision_3 = x_train[:, perm_important_features_1_precision[-
simplified_perm_x_train_recall_3 = x_train[:, perm_important_features_1_recall[-numDes
print("Results for 3 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_1_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_1_accuracy))
sum_3_important_features_perm_accuracy = np.sum(perm_importances_1_accuracy[perm_impor
print("The fraction of importance covered by the 3 features is ", sum_3_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_1_precision[-numDes
print("The total importance is ", np.sum(perm_importances_1_precision))
sum_3_important_features_perm_precision = np.sum(perm_importances_1_precision[perm_imp
print("The fraction of importance covered by the 3 features is ", sum_3_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_1_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_1_recall))
sum_3_important_features_perm_recall = np.sum(perm_importances_1_recall[perm_important
print("The fraction of importance covered by the 3 features is ", sum_3_important_feat
```

```
Results for 3 most important features
The importances of the most important  3  features for accuracy are  [0.0279665  0.09
151966 0.19200756]
These features are  Index(['qty_dot_domain', 'time_domain_activation', 'directory_len
gth'], dtype='object')
The total importance is  0.4340059506183266
The fraction of importance covered by the 3 features is  0.7177176201491984
The importances of the most important  3  features for precision are  [0.04802614 0.0
9374938 0.29131681]
These features are  Index(['qty_dot_domain', 'time_domain_activation', 'directory_len
gth'], dtype='object')
The total importance is  0.6342580139116339
The fraction of importance covered by the 3 features is  0.6828330424342659
The importances of the most important  3  features for recall are  [0.03200995 0.1797
0896 0.24449109]
These features are  Index(['qty_dot_domain', 'time_domain_activation', 'directory_len
gth'], dtype='object')
The total importance is  0.6075857008926758
The fraction of importance covered by the 3 features is  0.7508570431842418
```

In [ ]:

```
# Now, extracting the most important 4 features and points from x_train on normal Rand
numDesiredFeatures = 4
simplified_perm_x_train_accuracy_4 = x_train[:, perm_important_features_1_accuracy[-nu
simplified_perm_x_train_precision_4 = x_train[:, perm_important_features_1_precision[-
simplified_perm_x_train_recall_4 = x_train[:, perm_important_features_1_recall[-numDes
print("Results for 4 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_1_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_1_accuracy))
sum_4_important_features_perm_accuracy = np.sum(perm_importances_1_accuracy[perm_impor
print("The fraction of importance covered by the 4 features is ", sum_4_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_1_precision[-numDes
print("The total importance is ", np.sum(perm_importances_1_precision))
sum_4_important_features_perm_precision = np.sum(perm_importances_1_precision[perm_imp
print("The fraction of importance covered by the 4 features is ", sum_4_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_1_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_1_recall))
sum_4_important_features_perm_recall = np.sum(perm_importances_1_recall[perm_important
print("The fraction of importance covered by the 4 features is ", sum_4_important_feat
```

```
Results for 4 most important features
The importances of the most important  4  features for accuracy are  [0.02252915 0.02
79665  0.09151966 0.19200756]
These features are  Index(['length_url', 'qty_dot_domain', 'time_domain_activation',
       'directory_length'],
      dtype='object')
The total importance is  0.4340059506183266
The fraction of importance covered by the 4 features is  0.7696274010344947
The importances of the most important  4  features for precision are  [0.03726599 0.0
4802614 0.09374938 0.29131681]
These features are  Index(['length_url', 'qty_dot_domain', 'time_domain_activation',
       'directory_length'],
      dtype='object')
The total importance is  0.6342580139116339
The fraction of importance covered by the 4 features is  0.7415883018313058
The importances of the most important  4  features for recall are  [0.02748135 0.0320
0995 0.17970896 0.24449109]
These features are  Index(['length_url', 'qty_dot_domain', 'time_domain_activation',
       'directory_length'],
      dtype='object')
The total importance is  0.6075857008926758
The fraction of importance covered by the 4 features is  0.796087455302933
```

In [ ]:

```
# Now, extracting the most important 5 features and points from x_train on normal Ranc
numDesiredFeatures = 5
simplified_perm_x_train_accuracy_5 = x_train[:, perm_important_features_1_accuracy[-nu
simplified_perm_x_train_precision_5 = x_train[:, perm_important_features_1_precision[-
simplified_perm_x_train_recall_5 = x_train[:, perm_important_features_1_recall[-numDes
print("Results for 5 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_1_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_1_accuracy))
sum_5_important_features_perm_accuracy = np.sum(perm_importances_1_accuracy[perm_impor
print("The fraction of importance covered by the 5 features is ", sum_5_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_1_precision[-numDes
print("The total importance is ", np.sum(perm_importances_1_precision))
sum_5_important_features_perm_precision = np.sum(perm_importances_1_precision[perm_imp
print("The fraction of importance covered by the 5 features is ", sum_5_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_1_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_1_recall))
sum_5_important_features_perm_recall = np.sum(perm_importances_1_recall[perm_important
print("The fraction of importance covered by the 5 features is ", sum_5_important_feat
```

```
Results for 5 most important features
The importances of the most important  5  features for accuracy are  [0.01892917 0.02
252915 0.0279665  0.09151966 0.19200756]
These features are  Index(['asn_ip', 'length_url', 'qty_dot_domain', 'time_domain_act
ivation',
       'directory_length'],
      dtype='object')
The total importance is  0.4340059506183266
The fraction of importance covered by the 5 features is  0.8132424037636815
The importances of the most important  5  features for precision are  [0.03595594 0.0
3726599 0.04802614 0.09374938 0.29131681]
These features are  Index(['asn_ip', 'length_url', 'qty_dot_domain', 'time_domain_act
ivation',
       'directory_length'],
      dtype='object')
The total importance is  0.6342580139116339
The fraction of importance covered by the 5 features is  0.7982780637525662
The importances of the most important  5  features for recall are  [0.02108996 0.0274
8135 0.03200995 0.17970896 0.24449109]
These features are  Index(['ttl_hostname', 'length_url', 'qty_dot_domain',
       'time_domain_activation', 'directory_length'],
      dtype='object')
The total importance is  0.6075857008926758
The fraction of importance covered by the 5 features is  0.8307985428588672
```

In [ ]:

```
# Now, extracting the most important 2 features and points from x_train on improved Ra
numDesiredFeatures = 2
simplified_perm_x_train_accuracy_2 = x_train[:, perm_important_features_3_accuracy[-nu
simplified_perm_x_train_precision_2 = x_train[:, perm_important_features_3_precision[-
simplified_perm_x_train_recall_2 = x_train[:, perm_important_features_3_recall[-numDes
print("Results for 2 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_3_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_3_accuracy))
sum_2_important_features_perm_improved_accuracy = np.sum(perm_importances_3_accuracy[p
print("The fraction of importance covered by the 2 features is ", sum_2_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_3_precision[-numDes
print("The total importance is ", np.sum(perm_importances_3_precision))
sum_2_important_features_perm_improved_precision = np.sum(perm_importances_3_precision
print("The fraction of importance covered by the 2 features is ", sum_2_important_feat


print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_3_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_3_recall))
sum_2_important_features_perm_improved_recall = np.sum(perm_importances_3_recall[perm_
print("The fraction of importance covered by the 2 features is ", sum_2_important_feat
```

```
Results for 2 most important features
The importances of the most important  2  features for accuracy are  [0.08404191 0.18
566493]
These features are  Index(['time_domain_activation', 'directory_length'], dtype='obje
ct')
The total importance is  0.3972841490756761
The fraction of importance covered by the 2 features is  0.6788764188512959
The importances of the most important  2  features for precision are  [0.08481742 0.2
8078371]
These features are  Index(['time_domain_activation', 'directory_length'], dtype='obje
ct')
The total importance is  0.5789906225448995
The fraction of importance covered by the 2 features is  0.6314456892500111
The importances of the most important  2  features for recall are  [0.16819386 0.2346
2683]
These features are  Index(['time_domain_activation', 'directory_length'], dtype='obje
ct')
The total importance is  0.5542453022459535
The fraction of importance covered by the 2 features is  0.7267913482823809
```

In [ ]:

```
# Now, extracting the most important 3 features and points from x_train on improved Ra
numDesiredFeatures = 3
simplified_perm_x_train_accuracy_3 = x_train[:, perm_important_features_3_accuracy[-nu
simplified_perm_x_train_precision_3 = x_train[:, perm_important_features_3_precision[-
simplified_perm_x_train_recall_3 = x_train[:, perm_important_features_3_recall[-numDes
print("Results for 3 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_3_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_3_accuracy))
sum_3_important_features_perm_improved_accuracy = np.sum(perm_importances_3_accuracy[p
print("The fraction of importance covered by the 3 features is ", sum_3_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_3_precision[-numDes
print("The total importance is ", np.sum(perm_importances_3_precision))
sum_3_important_features_perm_improved_precision = np.sum(perm_importances_3_precision
print("The fraction of importance covered by the 3 features is ", sum_3_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_3_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_3_recall))
sum_3_important_features_perm_improved_recall = np.sum(perm_importances_3_recall[perm_
print("The fraction of importance covered by the 3 features is ", sum_3_important_feat
```

```
Results for 3 most important features
The importances of the most important  3  features for accuracy are  [0.02513643 0.08
404191 0.18566493]
These features are  Index(['qty_dot_domain', 'time_domain_activation', 'directory_len
gth'], dtype='object')
The total importance is  0.3972841490756761
The fraction of importance covered by the 3 features is  0.7421470707242865
The importances of the most important  3  features for precision are  [0.03859077 0.0
8481742 0.28078371]
These features are  Index(['qty_dot_domain', 'time_domain_activation', 'directory_len
gth'], dtype='object')
The total importance is  0.5789906225448995
The fraction of importance covered by the 3 features is  0.6980974904400311
The importances of the most important  3  features for recall are  [0.03363633 0.1681
9386 0.23462683]
These features are  Index(['qty_dot_domain', 'time_domain_activation', 'directory_len
gth'], dtype='object')
The total importance is  0.5542453022459535
The fraction of importance covered by the 3 features is  0.7874798673266029
```

In [ ]:

```
# Now, extracting the most important 4 features and points from x_train on improved Ra
numDesiredFeatures = 4
simplified_perm_x_train_accuracy_4 = x_train[:, perm_important_features_3_accuracy[-nu
simplified_perm_x_train_precision_4 = x_train[:, perm_important_features_3_precision[-
simplified_perm_x_train_recall_4 = x_train[:, perm_important_features_3_recall[-numDes
print("Results for 4 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_3_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_3_accuracy))
sum_4_important_features_perm_improved_accuracy = np.sum(perm_importances_3_accuracy[p
print("The fraction of importance covered by the 4 features is ", sum_4_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_3_precision[-numDes
print("The total importance is ", np.sum(perm_importances_3_precision))
sum_4_important_features_perm_improved_precision = np.sum(perm_importances_3_precision
print("The fraction of importance covered by the 4 features is ", sum_4_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_3_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_3_recall))
sum_4_important_features_perm_improved_recall = np.sum(perm_importances_3_recall[perm_
print("The fraction of importance covered by the 4 features is ", sum_4_important_feat
```

```
Results for 4 most important features
The importances of the most important  4  features for accuracy are  [0.0180958  0.02
513643 0.08404191 0.18566493]
These features are  Index(['length_url', 'qty_dot_domain', 'time_domain_activation',
       'directory_length'],
      dtype='object')
The total importance is  0.3972841490756761
The fraction of importance covered by the 4 features is  0.7876958351967474
The importances of the most important  4  features for precision are  [0.03195977 0.0
3859077 0.08481742 0.28078371]
These features are  Index(['asn_ip', 'qty_dot_domain', 'time_domain_activation',
       'directory_length'],
      dtype='object')
The total importance is  0.5789906225448995
The fraction of importance covered by the 4 features is  0.7532966028537355
The importances of the most important  4  features for recall are  [0.02352342 0.0336
3633 0.16819386 0.23462683]
These features are  Index(['length_url', 'qty_dot_domain', 'time_domain_activation',
       'directory_length'],
      dtype='object')
The total importance is  0.5542453022459535
The fraction of importance covered by the 4 features is  0.8299221168908547
```

In [ ]:

```
# Now, extracting the most important 5 features and points from x_train on improved Ra
numDesiredFeatures = 5
simplified_perm_x_train_accuracy_5 = x_train[:, perm_important_features_3_accuracy[-nu
simplified_perm_x_train_precision_5 = x_train[:, perm_important_features_3_precision[-
simplified_perm_x_train_recall_5 = x_train[:, perm_important_features_3_recall[-numDes
print("Results for 5 most important features")
print("The importances of the most important ", numDesiredFeatures, " features for acc
print("These features are ", feature_names[perm_important_features_3_accuracy[-numDesi
print("The total importance is ", np.sum(perm_importances_3_accuracy))
sum_5_important_features_perm_improved_accuracy = np.sum(perm_importances_3_accuracy[p
print("The fraction of importance covered by the 5 features is ", sum_5_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for pre
print("These features are ", feature_names[perm_important_features_3_precision[-numDes
print("The total importance is ", np.sum(perm_importances_3_precision))
sum_5_important_features_perm_improved_precision = np.sum(perm_importances_3_precision
print("The fraction of importance covered by the 5 features is ", sum_5_important_feat

print("The importances of the most important ", numDesiredFeatures, " features for rec
print("These features are ", feature_names[perm_important_features_3_recall[-numDesire
print("The total importance is ", np.sum(perm_importances_3_recall))
sum_5_important_features_perm_improved_recall = np.sum(perm_importances_3_recall[perm_
print("The fraction of importance covered by the 5 features is ", sum_5_important_feat
```

```
Results for 5 most important features
The importances of the most important  5  features for accuracy are  [0.01555198 0.01
80958  0.02513643 0.08404191 0.18566493]
These features are  Index(['asn_ip', 'length_url', 'qty_dot_domain', 'time_domain_act
ivation',
       'directory_length'],
      dtype='object')
The total importance is  0.3972841490756761
The fraction of importance covered by the 5 features is  0.8268415784654121
The importances of the most important  5  features for precision are  [0.02843746 0.0
3195977 0.03859077 0.08481742 0.28078371]
These features are  Index(['length_url', 'asn_ip', 'qty_dot_domain', 'time_domain_act
ivation',
       'directory_length'],
      dtype='object')
The total importance is  0.5789906225448995
The fraction of importance covered by the 5 features is  0.8024121904826714
The importances of the most important  5  features for recall are  [0.01332491 0.0235
2342 0.03363633 0.16819386 0.23462683]
These features are  Index(['time_response', 'length_url', 'qty_dot_domain',
       'time_domain_activation', 'directory_length'],
      dtype='object')
The total importance is  0.5542453022459535
The fraction of importance covered by the 5 features is  0.8539636545490663
```

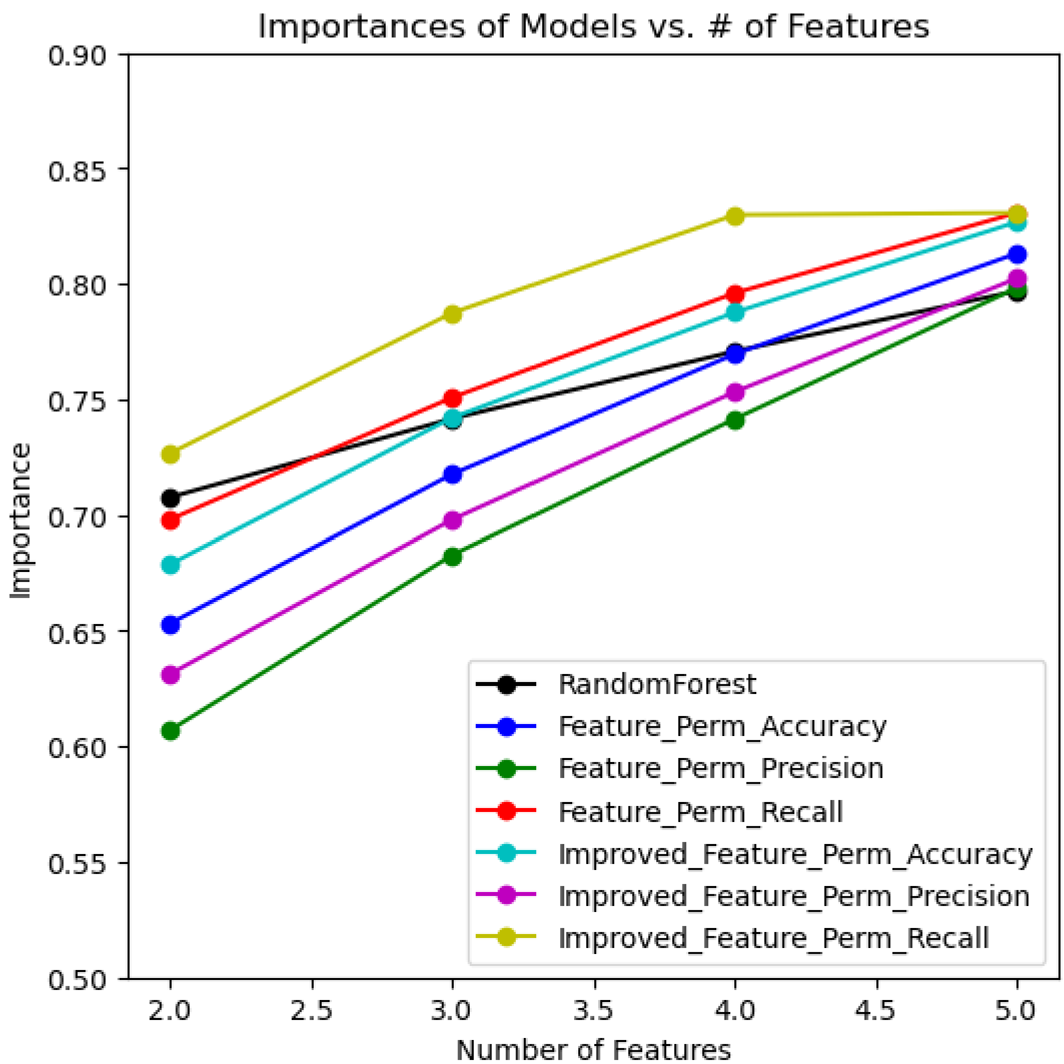
Visualization

Below is a line graph revealing the importance captured by the model features as a function of the number of features. As the number of features increases, the total importance captured increases a lot at first, but then begins to plateau. After observing the graph, our group decided to work with 4 features. For the RandomForest, the importance is a measure of Gini impurity. The importances of the six feature permutation models are based on accuracy, precision, and recall. Three of the feature permutation models utilized the original RandomForest while the other three feature permutations utilized an improved RandomForest (the minimum number of points in a leaf is increased to 5 to reduce overfitting).

In [ ]:

```
random_forest_accuracies = [sum_2_important_features_forest, sum_3_important_features_
perm_accuracies = [sum_2_important_features_perm_accuracy, sum_3_important_features_pe
perm_precisions = [sum_2_important_features_perm_precision, sum_3_important_features_p
perm_recalls = [sum_2_important_features_perm_recall, sum_3_important_features_perm_re
perm_improved_accuracies = [sum_2_important_features_perm_improved_accuracy, sum_3_imp
perm_improved_precisions = [sum_2_important_features_perm_improved_precision, sum_3_in
perm_improved_recalls = [sum_2_important_features_perm_improved_recall, sum_3_importar

x = [2, 3, 4, 5]
plt.plot(x, random_forest_accuracies, 'ko-')
plt.plot(x, perm_accuracies, 'bo-')
plt.plot(x, perm_precisions, 'go-')
plt.plot(x, perm_recalls, 'ro-')
plt.plot(x, perm_improved_accuracies, 'co-')
plt.plot(x, perm_improved_precisions, 'mo-')
plt.plot(x, perm_improved_recalls, 'yo-')

plt.legend(['RandomForest', 'Feature_Perm_Accuracy', 'Feature_Perm_Precision', 'Featur

plt.title("Importances of Models vs. # of Features")
```

```
plt.ylabel("Importance")
plt.xlabel("Number of Features")
(plt.gcf()).set_figheight(6)
plt.gcf().set_figwidth(6)
plt.gca().set_ylim(0.5, 0.9)

plt.show()
numDesiredFeatures = 4 # Resetting back to 4 because that's what we use for our models
```

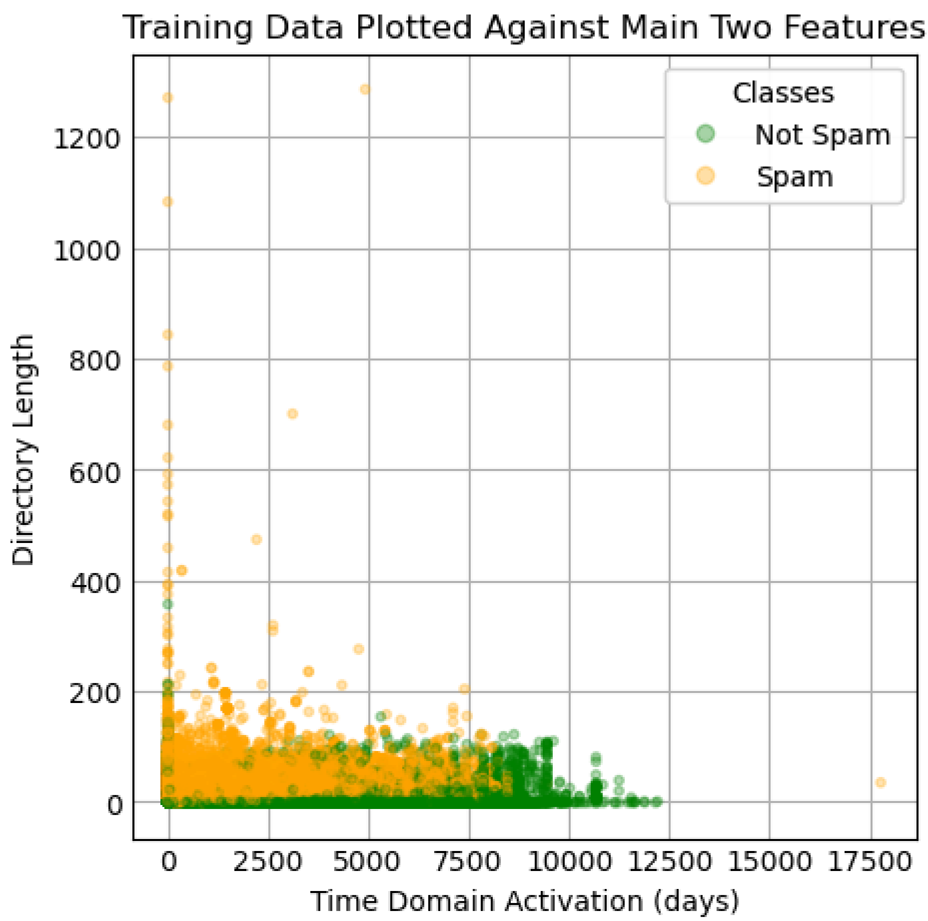


Since both feature permutation and impurity-based feature importance indicate "directory_length" and "time_domain_activation" as the 2 most important features, we will try to visualize the model by only focusing on these two parameters for now. Below is a scatter plot which shows all the training set data points plotted against these two features.

In [ ]:

```
from matplotlib.colors import LinearSegmentedColormap
import matplotlib.scale as scale

plt.style.use('_mpl-gallery')
fig1, ax1 = plt.subplots(figsize=(4,4))
# fig2, ax2 = plt.subplots(figsize=(4,4))
# fig3, ax3 = plt.subplots(figsize=(4,4))
custom_colormap = LinearSegmentedColormap.from_list("viridis", colors=[(0,"green"), (1

scatter1 = ax1.scatter(simplified_perm_x_train_accuracy_2[:, 0], simplified_perm_x_tra
```

```
handles, labels = scatter1.legend_elements(prop="colors")
legend1 = ax1.legend(handles, ["Not Spam", "Spam"], loc="upper right", title="Classes'
ax1.add_artist(legend1)
# ax1.set_xscale("log")
# ax1.set_yscale("log")
# ax1 = scale.LogScale(axis=ax1, base=10)
# fig1.suptitle("Training Data Plotted Against Main Two Features")
# fig1.supxlabel("Directory Length")
# fig1.supylabel("Time Domain Activation")


"""
scatter2 = ax2.scatter(simplified_perm_x_train_precision_2[:, 0], simplified_perm_x_tr
handles, labels = scatter2.legend_elements(prop="colors")
legend2 = ax2.legend(handles, ["Not Spam", "Spam"], loc="upper right", title="Classes'
ax2.add_artist(legend2)
fig2.suptitle("Training Data Plotted Against Main Two Features")
fig2.supxlabel("Directory Length")
fig2.supylabel("Time Domain Activation")

scatter3 = ax3.scatter(simplified_perm_x_train_recall_2[:, 0], simplified_perm_x_trair
handles, labels = scatter3.legend_elements(prop="colors")
legend3 = ax3.legend(handles, ["Not Spam", "Spam"], loc="upper right", title="Classes'
ax3.add_artist(legend3)
fig3.suptitle("Training Data Plotted Against Main Two Features")
fig3.supxlabel("Directory Length")
fig3.supylabel("Time Domain Activation")
"""
plt.title("Training Data Plotted Against Main Two Features")
plt.ylabel("Directory Length")
plt.xlabel("Time Domain Activation (days)")

plt.show()
```



2. Training using Naive Bayes Approach

In this part we attempt to train our preprocessed dataset with the Gaussian Naive Bayes model and use external evaluation methods to evaluate our results. All the feature permutation models used for feature selection with the exception of one ranked "Directory Length", "Time Domain Activation", "Qty Dot Domain", and "Length URL" as the four most important features; the other model swapped "ASN IP" for "Length URL". Therefore, the pre-processed training data respresenting feature permutation will contain the majority 4 important features.

In [ ]:

```
#importing the library from scikit
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB() #making the training object
```

In [ ]:

```
def evaluate(predicted, actual):
    num_equal = np.count_nonzero(predicted == actual)
    true_positive = 0
    true_negative = 0
    false_negative = 0
    false_positive = 0
    for i in range(len(predicted)):
        if predicted[i] == 1 and actual[i] == 1:
            true_positive += 1
        elif predicted[i] == 1 and actual[i] == 0:
            false_positive += 1
        elif predicted[i] == 0 and actual[i] == 1:
```

```
            false_negative += 1
        else:
            true_negative += 1
    print("# of correctly labeled points is: " + str(num_equal))
    accuracy = str(num_equal/len(actual))
    print("% of accuracy is "+str(accuracy))
    recall = true_positive / (true_positive + false_negative)
    print("recall is: " + str(recall))
    precision = true_positive / (true_positive + false_positive)
    print("precision is: " + str(precision))
    f1_measure = (2*precision*recall)/(precision+recall)
    print("the f1 score is: " + str(f1_measure))
    return np.array([accuracy, recall, precision, f1_measure], dtype=float)
```

With Training Data

Now we will train our dataset with the previous simplified dataset we got from the feature permutations, and the four most desired features. We then predict the labels for the same dataset put in to see the accuracy.

In [ ]:
```
#training the data set with simplified_perm_x_train_4, the training data,
#and y_train, the labels for each of those data points
y_pred_perm = gnb.fit(simplified_perm_x_train_recall_4, y_train).predict(simplified_pe
#fit -> learn patterns
#predict -> used learned patterns to predict labels for dataset

#evaluation
gnb_train_perm = evaluate(y_pred_perm, y_train)
```

```
# of correctly labeled points is: 59864
% of accuracy is 0.8441417431645445
recall is: 0.6193330071754729
precision is: 0.8985035783994795
the f1 score is: 0.7332448413177265
<class 'numpy.ndarray'>
```

If we try with the tree based selected features, we get the following results.

In [ ]:
```
y_pred_tree = gnb.fit(simplified_x_train_4, y_train).predict(simplified_x_train_4)
#evaluation
gnb_train_forest = evaluate(y_pred_tree, y_train)
```

```
# of correctly labeled points is: 58241
% of accuracy is 0.8212558342851503
recall is: 0.5909165035877365
precision is: 0.8458216619981326
the f1 score is: 0.6957565284178188
```

With Test Data

Same as above, but we predict the labels for the testing dataset made in the beginning to see the accuracy.

In [ ]:
```
#training the data set with simplified_perm_x_train_4, the training data,
#and y_train, the labels for each of those data points
y_pred_perm_test = gnb.fit(simplified_perm_x_train_recall_4, y_train).predict(x_test[:
```

```
#fit -> learn patterns
#predict -> used learned patterns to predict labels for dataset

#evaluation
gnb_test_perm = evaluate(y_pred_perm_test, y_test)
```

```
# of correctly labeled points is: 14963
% of accuracy is 0.8439368302312464
recall is: 0.6170942964536689
precision is: 0.8990476190476191
the f1 score is: 0.7318538618083149
```

In [ ]:

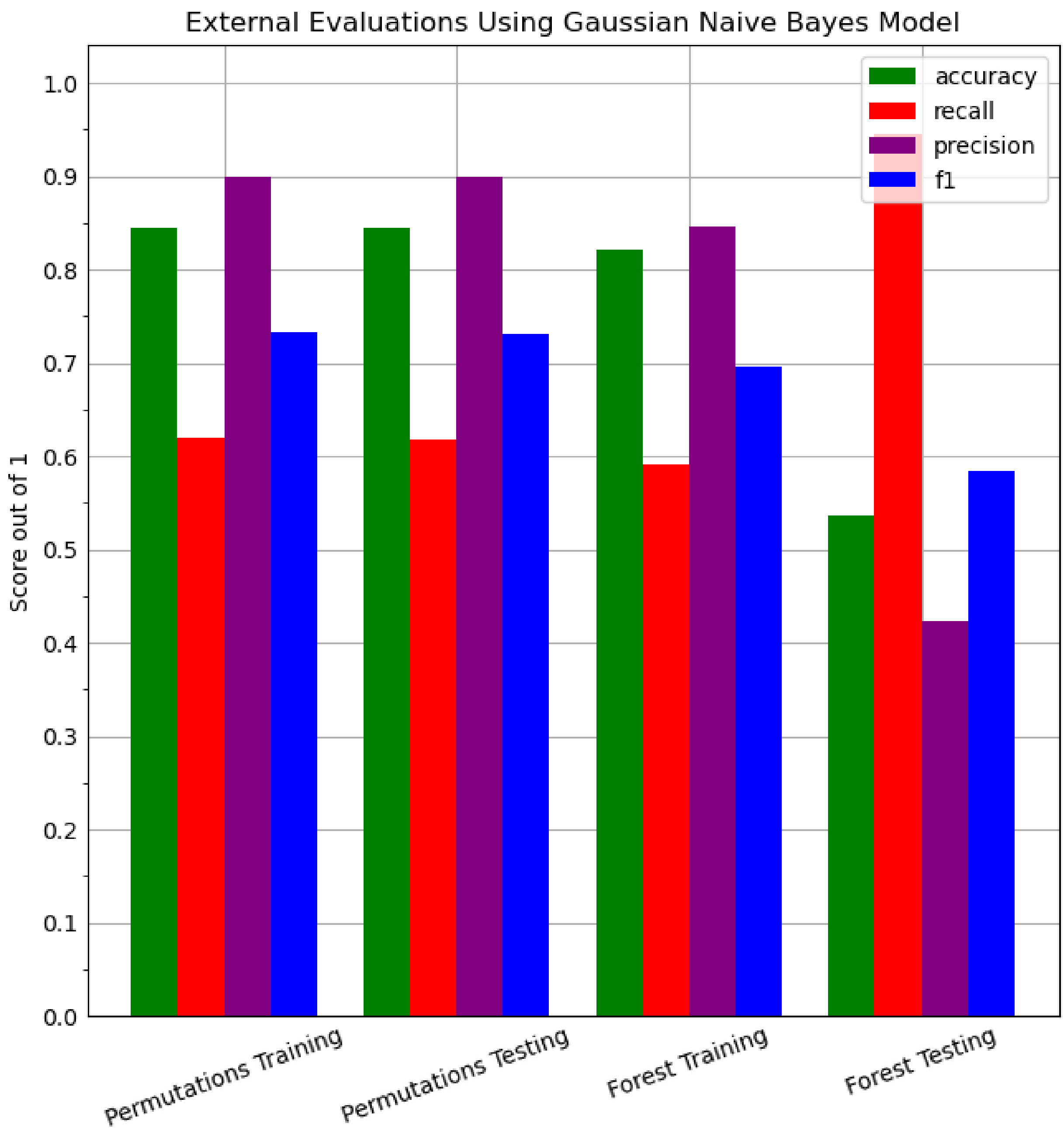
```
y_pred_tree_test = gnb.fit(simplified_x_train_4, y_train).predict(x_test[:, perm_impor

#evaluation
gnb_test_forest = evaluate(y_pred_tree_test, y_test)
# print(type(gnb_test_forest))
```

```
# of correctly labeled points is: 9499
% of accuracy is 0.5357586012408347
recall is: 0.9459061938225201
precision is: 0.4228521332554062
the f1 score is: 0.5844398444994194
```

Visualization

In [ ]:

```
fig, ax = plt.subplots()
fig.set_figheight(6)
fig.set_figwidth(6)
width = 0.25

offset = np.array([0, width, 2*width, 3*width])
multiplier = width * 5
labels = ["accuracy", "recall", "precision", "f1"]
rects = ax.bar(x=offset, height=gnb_train_perm, width=width, color=["green", "red", "p
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier, height=gnb_test_perm, width=width, color=["green",
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier*2, height=gnb_train_forest, width=width, color=["gr
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier*3, height=gnb_test_forest, width=width, color=["gre
# ax.bar_label(rects, padding=3)

ax.set_ylabel("Score out of 1")
ax.set_title("External Evaluations Using Gaussian Naive Bayes Model")
labels = ["Permutations Training", "Permutations Testing", "Forest Training", "Forest
ax.set_xticks((1.5*width) + np.array([0, multiplier, multiplier*2, multiplier*3]), lab
ax.set_xticklabels(labels, rotation=20)
ax.set_ymargin(0.1)
ax.set_yticks(ticks=0.05*np.array(range(21)), minor=True)
ax.set_yticks(ticks=0.1*np.array(range(11)))
ax.legend(loc='upper right')

plt.show()
```



3. Training using Random Forests

Here, we attempt to use random forests to train our data. These basically take random data points and then based on the number of selected data points they grow decision trees with more sample data points until each "branch" is pure, or they are leaves.

In [ ]:
```
#importing the library from scikit
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier(n_estimators=10, bootstrap=True)
```

With Training Data

Here, we made a RandomForestClassifier object with 10 trees, no max_depth, and where bootstrapping is True. We are predicting labels for the same dataset we put in to train.

```
In [ ]: y_pred_perm = clf.fit(simplified_perm_x_train_recall_4, y_train).predict(simplified_pe

        #evaluation
        rf_train_perm = evaluate(y_pred_perm, y_train)
```

```
# of correctly labeled points is: 68947
% of accuracy is 0.9722210471396139
recall is: 0.9670172863666014
precision is: 0.9533360128617363
the f1 score is: 0.960127914507772
```

As seen above, this method has a higher accuracy in internal evaluation. If we try with the tree based features, the results are as follows:

```
In [ ]: y_pred_tree = clf.fit(simplified_x_train_4, y_train).predict(simplified_x_train_4)

        #evaluation
        rf_train_forest = evaluate(y_pred_tree, y_train)
```

```
# of correctly labeled points is: 70488
% of accuracy is 0.9939506747324337
recall is: 0.9878506196999348
precision is: 0.994622552440376
the f1 score is: 0.9912250199431365
```

With Testing Data

We are now doing the same but predicting the labels for the testing data.

```
In [ ]: y_pred_test_perm = clf.fit(simplified_perm_x_train_recall_4, y_train).predict(x_test[:

        #evaluation
        rf_test_perm = evaluate(y_pred_test_perm, y_test)
```

```
# of correctly labeled points is: 16666
% of accuracy is 0.9399887196841512
recall is: 0.9182873018467069
precision is: 0.9087821445900048
the f1 score is: 0.913509998374248
```

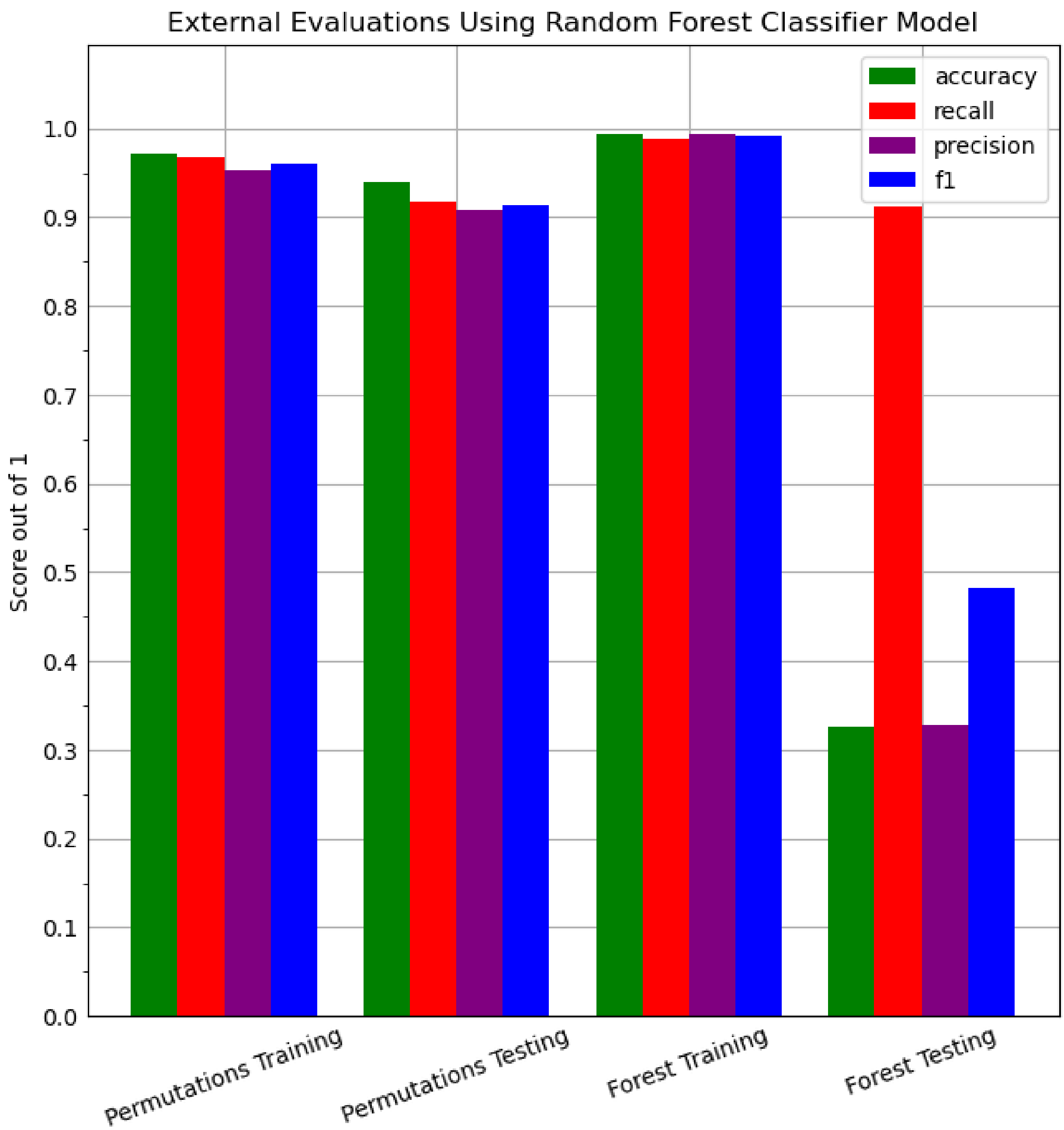
```
In [ ]: y_pred_test_tree = clf.fit(simplified_x_train_4, y_train).predict(x_test[:, perm_impor

        #evaluation
        rf_test_forest = evaluate(y_pred_test_tree, y_test)
```

```
# of correctly labeled points is: 5772
% of accuracy is 0.3255499153976311
recall is: 0.9117502859944435
precision is: 0.32823439430487733
the f1 score is: 0.48269596816058136
```

Visualization

In [ ]:

```
fig, ax = plt.subplots()
fig.set_figheight(6)
fig.set_figwidth(6)
width = 0.25

offset = np.array([0, width, 2*width, 3*width])
multiplier = width * 5
labels = ["accuracy", "recall", "precision", "f1"]
rects = ax.bar(x=offset, height=rf_train_perm, width=width, color=["green", "red", "pu
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier, height=rf_test_perm, width=width, color=["green",
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier*2, height=rf_train_forest, width=width, color=["gre
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier*3, height=rf_test_forest, width=width, color=["gree
# ax.bar_label(rects, padding=3)

ax.set_ylabel("Score out of 1")
ax.set_title("External Evaluations Using Random Forest Classifier Model")
labels = ["Permutations Training", "Permutations Testing", "Forest Training", "Forest
ax.set_xticks((1.5*width) + np.array([0, multiplier, multiplier*2, multiplier*3]), lab
ax.set_xticklabels(labels, rotation=20)
ax.set_ymargin(0.1)
ax.set_yticks(ticks=0.05*np.array(range(21)), minor=True)
ax.set_yticks(ticks=0.1*np.array(range(11)))
ax.legend(loc='upper right')

plt.show()
```



4. Training Using Logistic Regression

Here we attempt to create a model using logistic regression. Logistic regression uses the sigmoid function ($\frac{1}{1+exp(-x)}$) to perform its binary classification. The sigmoid function's range is values bettween 0 and 1. So, when the data is inputted, a soft assignment is created with proabalities of which class it belongs to (i.e 0 or 1). If the proablity is closer to 1 then the binary classification is 1 or 0 otherwise. Below, we used the default 0.5 as our threshold without any regularization.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='liblinear', random_state=0)
```

With Training Data

Here, we fit a model using the training data of the feature permuation dimension reduction as well as the tree based feature selection. The results are given below

In [ ]:
```
y_hat_perm = model.fit(simplified_perm_x_train_recall_4, y_train).predict(simplified_p
lr_train_perm = evaluate(y_hat_perm, y_train)
```

```
# of correctly labeled points is: 63091
% of accuracy is 0.8896456420886388
recall is: 0.7629084103023152
precision is: 0.9036580076148248
the f1 score is: 0.827339716718881
```

In [ ]:
```
y_hat_tree = model.fit(simplified_x_train_4, y_train).predict(simplified_x_train_4)
lr_train_forest = evaluate(y_hat_tree, y_train)
```

```
# of correctly labeled points is: 62646
% of accuracy is 0.8833707009602775
recall is: 0.7545265899011271
precision is: 0.8923106534501011
the f1 score is: 0.8176547102008421
```

With Testing Data

Here, we fit the same models using the training data of the feature permuations and tree based feature selection. However, we predict and test accuracy using the testing data that was not used in the model generation. The results are shown below

In [ ]:
```
y_hat_perm_test = model.fit(simplified_perm_x_train_recall_4, y_train)
x_test_values = x_test[:,perm_important_features_1_recall[-numDesiredFeatures: :]]
y_hat_perm_test = y_hat_perm_test.predict(x_test_values)
lr_test_perm = evaluate(y_hat_perm_test, y_test)
```

```
# of correctly labeled points is: 15758
% of accuracy is 0.8887760857304005
recall is: 0.7632619439868205
precision is: 0.8964783281733746
the f1 score is: 0.8245239366435309
```

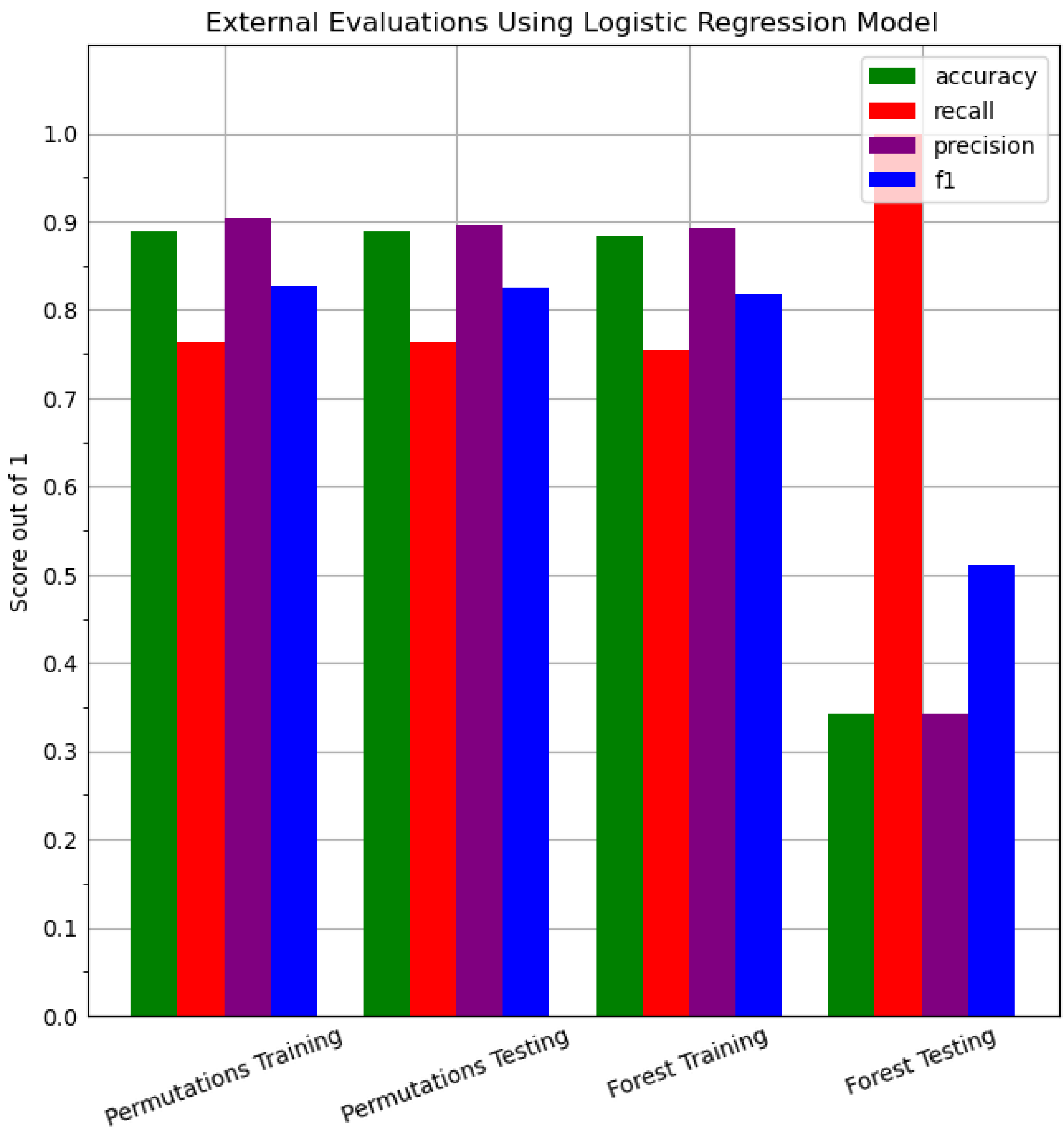
In [ ]:
```
y_hat_tree_test = model.fit(simplified_x_train_4, y_train)
x_test_values = x_test[:,perm_important_features_1_recall[-numDesiredFeatures: :]]
y_hat_tree_test = y_hat_tree_test.predict(x_test_values)
lr_test_forest = evaluate(y_hat_tree_test, y_test)
```

```
# of correctly labeled points is: 6073
% of accuracy is 0.34252679075014103
recall is: 1.0
precision is: 0.3424155243414001
the f1 score is: 0.5101483380258016
```

Visualization

In [ ]:
```
fig, ax = plt.subplots()
fig.set_figheight(6)
```

```
fig.set_figwidth(6)
width = 0.25

offset = np.array([0, width, 2*width, 3*width])
multiplier = width * 5
labels = ["accuracy", "recall", "precision", "f1"]
rects = ax.bar(x=offset, height=lr_train_perm, width=width, color=["green", "red", "pu
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier, height=lr_test_perm, width=width, color=["green",
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier*2, height=lr_train_forest, width=width, color=["gre
# ax.bar_label(rects, padding=3)

rects = ax.bar(x=offset+multiplier*3, height=lr_test_forest, width=width, color=["gree
# ax.bar_label(rects, padding=3)

ax.set_ylabel("Score out of 1")
ax.set_title("External Evaluations Using Logistic Regression Model")
labels = ["Permutations Training", "Permutations Testing", "Forest Training", "Forest
ax.set_xticks((1.5*width) + np.array([0, multiplier, multiplier*2, multiplier*3]), lab
ax.set_xticklabels(labels, rotation=20)
ax.set_ymargin(0.1)
ax.set_yticks(ticks=0.05*np.array(range(21)), minor=True)
ax.set_yticks(ticks=0.1*np.array(range(11)))
ax.legend(loc='upper right')

plt.show()
```



Appendix

Principal Component Analysis

Using SVD Decomposition of the centered data, the V matrix will provide the eigenvectors which are the components in the $Zspace.\frac{\Sigma^2}{n}$ matrix will provide the eigenvalues, a measure of how impactful the eigenvectors are. For now, we will only keep the components/ features which cover 95% of the variance in the data.

In [ ]:

```
feature_means = np.mean(x, axis=0)
# feature_stdv = np.std(x, axis=0)
x_standardized = (x-feature_means) # / feature_stdv
print("Shape of x_standardized is ", np.shape(x_standardized))
```

```
#################################################################################
# THE BELOW LINE WILL USE ALL AVAILABLE RAM ON COLAB
# I DOWNLOADED THE .ipynb FILE AND OPENED IT ON VS CODE
#################################################################################
U, S, Vt = np.linalg.svd(x_standardized, full_matrices=False) # full_matrices=False sh
print("Finished svd Decomposition")

numFeatures = np.shape(Vt)[0]
eigenvalues = S**2 / np.shape(x_standardized)[0] # (D, )
features_ascending_order = np.argsort(eigenvalues)

print("Shape of the eigenvalues is ", np.shape(eigenvalues))
numDesiredFeatures = 2
desiredFeatures = feature_names[features_ascending_order[-1*numDesiredFeatures: :]]
importantData = np.hstack((x_standardized[:, features_ascending_order[-1*numDesiredFea

accuracy = np.sum(eigenvalues[features_ascending_order[-1*(numDesiredFeatures+1): :]])
print("Accuracy is ", accuracy, "from ", numDesiredFeatures, " features.")
print("These features are ", desiredFeatures)
```

```
Shape of x_standardized is  (88647, 111)
Finished svd Decomposition
Shape of the eigenvalues is  (111,)
Accuracy is  0.9998616072116091 from  2  features.
These features are  Index(['qty_hyphen_url', 'qty_dot_url'], dtype='object')
```