Zachary Panzarino
zpanzarino3

Introduction to the Dataset

The aim of this project was to utilize different features of a website in order to predict if that website is a phishing website. Phishing refers to the fraudulent practice of impersonating a website from a reputable company to steal user information such as login information or credit card numbers. This type of attack exploits the weakest part of a security network, the human behind the computer, as not everyone is able to determine if a website is fraudulent. Therefore, a tool that utilizes information about a website in order to determine if it is fraudulent or not could be important in improving an organization's human firewall. This is important to myself since I personally have experience working in this industry. In the past I have worked at a small company called PhishTrain, where we would send fake phishing emails to a companies' employees in order to test their phishing awareness. If an employee fell victim to the phishing attack, they were placed into a training program to learn about phishing attacks. However, a tool that could automatically recognize phishing websites would be of great use as it could be marketed as another product that could greatly cut down on individuals falling victim to these types of attacks.

The dataset used for this study consists of 11055 different datapoints with 30 different features each. The features were trinary with possible values of common to phishing websites, suspicious, or commonly legitimate. Some features were binary and only included phishing or legitimate, without the suspicious option. Overall, the features encompassed information about the website address, links contained on the page, HTML and JavaScript features, and domain registration/ranking information. A full description of each of the features can be found in the included `features.pdf` file. The output of each datapoint was binary, whether the website was actually a phishing attack or a legitimate site. The core metrics that will be used to measure performance on the dataset are weighted precision and mean accuracy. These metrics essentially show in what percentage of trials the algorithm was able to predict the correct result.

Description of the Algorithms

Since this dataset involves classification, the first algorithm that will be used is the random forests with boosting. The random forests algorithm consists of creating a series of decision tree classifiers on various samples of the dataset. The algorithm utilizes averaging in order to improve the predictive accuracy while controlling for over-fitting. The first hyperparameter that needs to be tuned is the number of estimators, which refers to the number of trees in the random forest. The next is the maximum depth of the tree, which is the limit for the total height of an individual tree. A higher number on these hyperparameters generally correlates with a higher accuracy on training data and higher chance of overfitting. The last two hyperparameters to optimize are the minimum number of samples to split an internal node and the minimum number of samples to be at a leaf node, which both control the structure of an individual tree.

The next algorithm that will be used to classify the data is a support vector machine. A support vector machine determines an optimal hyperplane that categorizes examples. The first

hyperparameter that can be tuned is the C value, which is the penalty parameter which controls the smoothness of the decision boundary versus the correctness of point classification. The next hyperparameter is gamma, which controls how accurately the algorithm attempts fit the training data. If this value is too low it may not match the data well, but if it is too high then it will cause overfitting. The type of kernel decides what type of hyperplane is used to separate the data. In this case the kernel can be one of a couple different non-linear hyperplanes. Finally, the last hyperparameter is probability which determines whether or not to use probability estimates.

Finally, the last algorithm that is being used is a neural network with at least two layers. The first hyperparameter is the size of hidden layers, which determines the number of neurons in each hidden layer and the number of hidden layers. The next is the activation function, which determines when a neuron should fire. Another hyperparameter is the solver function, which determines which solver should be used to optimize the weights. The last hyperparameter is alpha, which refers to the penalty/regularization term.

Tuning Hyperparameters

In order to tune the hyperparameters, a cross-validated grid search was used to find the most optimal hyperparameters. This function is used to test all possible hyperparameters in order to find the ones that work best with the training data. For reference, all machine learning estimators have been implemented through scikit-learn.

The first function to optimize was the random forests. To start, a model was trained with the default hyperparameters for reference. The results are shown below (the number underneath "weighted avg" is the respective accuracy score.

```
Train
              precision    recall  f1-score   support

          -1     0.9938    0.9849    0.9894      3917
           1     0.9881    0.9951    0.9916      4927

    accuracy                         0.9906      8844
   macro avg     0.9910    0.9900    0.9905      8844
weighted avg     0.9906    0.9906    0.9906      8844

0.9906151062867481

Test
              precision    recall  f1-score   support

          -1     0.9477    0.9235    0.9355       981
           1     0.9402    0.9593    0.9497      1230

    accuracy                         0.9435      2211
   macro avg     0.9440    0.9414    0.9426      2211
weighted avg     0.9435    0.9435    0.9434      2211

0.9434644957033017
```

Even without any hyper tuning, the model is quite accurate. For the first round of hyper tuning, the following hyperparameter options were chosen. A wide range of number of estimators and max depth was chosen in order to support a large variety of options.

```
{
    'n_estimators': [100, 400, 700, 1000, 1300, 1700, 2000],
    'max_depth': [None, 20, 40, 60, 80, 100],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
```

When the grid search was run while fitting with 3 folds, the performance results for the optimal hyperparameters were as follows (the optimal hyperparameters are listed at the bottom). The search took 7.2 minutes to complete.

```
Train
              precision    recall  f1-score   support

          -1     0.9920    0.9867    0.9894      3917
           1     0.9895    0.9937    0.9916      4927

    accuracy                         0.9906      8844
   macro avg     0.9908    0.9902    0.9905      8844
weighted avg     0.9906    0.9906    0.9906      8844

0.9906151062867481

Test
              precision    recall  f1-score   support

          -1     0.9478    0.9256    0.9366       981
           1     0.9417    0.9593    0.9505      1230

    accuracy                         0.9444      2211
   macro avg     0.9448    0.9425    0.9435      2211
weighted avg     0.9444    0.9444    0.9443      2211

0.9443690637720489

{'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'n_estimators': 700}
```

While the metrics for the testing set suffered a bit, the training set was matched more accurately. In addition, this provided a good idea of what the ideal hyperparameters would be. Since the minimum samples for a leaf and the minimum samples to split were the same as the default, they were eliminated for the second round of tuning. The second grid search was run with the following hyperparameter options.

```
params = {
    'n_estimators': [500, 600, 700, 800, 900],
    'max_depth': [None, 120, 140, 160, 180, 200, 220, 240]
}
```

This time the search was run with 5 folds and took 2.4 minutes, yielding the following results.

```
Train
              precision    recall  f1-score   support
```

```
          -1     0.9948    0.9839    0.9893      3917
           1     0.9873    0.9959    0.9916      4927

    accuracy                         0.9906      8844
   macro avg     0.9911    0.9899    0.9905      8844
weighted avg     0.9907    0.9906    0.9906      8844

0.9906151062867481

Test
              precision    recall  f1-score   support

          -1     0.9459    0.9266    0.9361       981
           1     0.9424    0.9577    0.9500      1230

    accuracy                         0.9439      2211
   macro avg     0.9441    0.9422    0.9431      2211
weighted avg     0.9439    0.9439    0.9439      2211

0.9439167797376753

{'max_depth': None, 'n_estimators': 700}
```

Overall, the performance metrics hardly changed even with the most optimal hyperparameters. This is most likely because the data is strongly correlated and the random forest works extremely well for the data. The optimal, non-default hyperparameters are 700 estimators with no maximum depth.

A similar process was performed to optimize the hyperparameters for the support vector machine. The results of the default case were as follows.

```
Train
              precision    recall  f1-score   support

          -1     0.9605    0.9377    0.9490      3917
           1     0.9514    0.9694    0.9603      4927

    accuracy                         0.9553      8844
   macro avg     0.9560    0.9535    0.9546      8844
weighted avg     0.9554    0.9553    0.9553      8844

0.9553369516056083

Test
              precision    recall  f1-score   support

          -1     0.9419    0.9093    0.9253       981
           1     0.9296    0.9553    0.9423      1230

    accuracy                         0.9349      2211
   macro avg     0.9358    0.9323    0.9338      2211
weighted avg     0.9351    0.9349    0.9347      2211

0.9348710990502035
```

Already, the performance is less than that of the random forests. However, some hyper tuning may increase the performance. The set of possible hyperparameters are as follows.

```
{
    'C': [0.1, 1, 10, 100, 1000],
    'gamma': ['auto', 'scale'],
    'kernel': ['rbf', 'poly', 'sigmoid'],
    'probability': [True, False]
}
```

The results of running these parameters with a cross-validated grid search with 5 folds are as follows. The search took 4 minutes to run.

```
Train
              precision    recall  f1-score   support

          -1     0.9902    0.9778    0.9839      3917
           1     0.9825    0.9923    0.9874      4927

    accuracy                         0.9859      8844
   macro avg     0.9863    0.9850    0.9857      8844
weighted avg     0.9859    0.9859    0.9859      8844

0.9858661239258254

Test
              precision    recall  f1-score   support

          -1     0.9138    0.9297    0.9217       981
           1     0.9431    0.9301    0.9366      1230

    accuracy                         0.9299      2211
   macro avg     0.9285    0.9299    0.9291      2211
weighted avg     0.9301    0.9299    0.9300      2211

0.9298959746720941

{'C': 100, 'gamma': 'scale', 'kernel': 'rbf', 'probability': True}
```

While the performance increases for the training set, it decreases for the testing set. This could possibly be a result of overfitting. However, since it improves the training set it might perform better with different testing sets. Therefore, the chosen optimal hyperparameters for this function are a C value of 100, scaled gamma, RBF as the kernel with probability estimates.

Finally, the neural network needed to be tuned. The performance results of a default neural network are as follows.

```
Train
              precision    recall  f1-score   support

          -1     0.9887    0.9847    0.9867      3917
           1     0.9879    0.9911    0.9895      4927

    accuracy                         0.9882      8844
   macro avg     0.9883    0.9879    0.9881      8844
weighted avg     0.9882    0.9882    0.9882      8844
```

```
0.9882406151062868

Test
              precision    recall  f1-score   support

          -1     0.9234    0.9337    0.9285       981
           1     0.9467    0.9382    0.9424      1230

    accuracy                         0.9362      2211
   macro avg     0.9350    0.9360    0.9355      2211
weighted avg     0.9363    0.9362    0.9363      2211

0.9362279511533242
```

Again, this model performs very well for the given data. A cross-validated grid search was performed with the following possible hyperparameters.

```
{
    'hidden_layer_sizes': [(100, 100), (200, 100), (200, 200)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['lbfgs', 'sgd', 'adam'],
    'alpha': [.001, .0001, .00001]
}
```

The results of running the search with 3 folds are as follows. The search took 8 minutes to run.

```
Train
              precision    recall  f1-score   support

          -1     0.9977    0.9770    0.9872      3917
           1     0.9820    0.9982    0.9900      4927

    accuracy                         0.9888      8844
   macro avg     0.9898    0.9876    0.9886      8844
weighted avg     0.9889    0.9888    0.9888      8844

0.9888059701492538

Test
              precision    recall  f1-score   support

          -1     0.9365    0.9174    0.9269       981
           1     0.9352    0.9504    0.9427      1230

    accuracy                         0.9358      2211
   macro avg     0.9359    0.9339    0.9348      2211
weighted avg     0.9358    0.9358    0.9357      2211

0.9357756671189507

{'activation': 'tanh', 'alpha': 1e-05, 'hidden_layer_sizes': (200, 200),
'solver': 'adam'}
```

There was almost no noticeable difference between this model and the default model. Therefore, another round of optimization may produce better results. For the next round, the activation function was stable at tanh and the solver was adam (the default). This round was used to focus

on hidden layer size optimization. Another cross-validated grid search was run with the following possible hyperparameters.

```
{
    'hidden_layer_sizes': [(200, 200), (300, 200), (300, 300), (300, 200,
100), (200, 200, 200)],
    'activation': ['tanh'],
    'alpha': [.01, .001, .0001]
}
```

The results of running this search with 3 folds are as follows. The search took 6.5 minutes to run.

```
Train
              precision    recall  f1-score   support

          -1     0.9895    0.9890    0.9893      3917
           1     0.9913    0.9917    0.9915      4927

    accuracy                         0.9905      8844
   macro avg     0.9904    0.9904    0.9904      8844
weighted avg     0.9905    0.9905    0.9905      8844

0.9905020352781547

Test
              precision    recall  f1-score   support

          -1     0.9037    0.9470    0.9248       981
           1     0.9560    0.9195    0.9374      1230

    accuracy                         0.9317      2211
   macro avg     0.9299    0.9333    0.9311      2211
weighted avg     0.9328    0.9317    0.9318      2211

0.9317051108095884

{'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (200, 200,
200)}
```

These results had the best performance for the training dataset with almost identical results for the testing dataset. Therefore, this model is most likely going to perform the best with other training datasets. The optimal hyperparameters for the neural network are a tanh activation function, an alpha value of 0.0001 and 3 hidden layers of size 200 each.

To prove that the data does not have a non-trivial distribution, linear regression model was trained and scored on the train and test data. The accuracy scores were as follows.

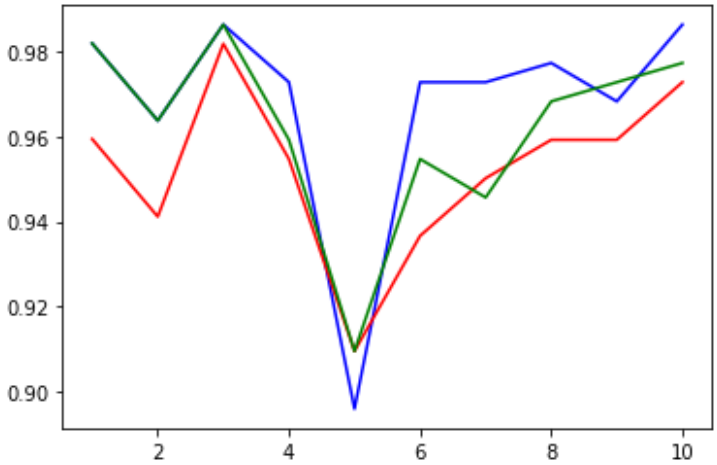
```
Train
0.7019712092395985
Test
0.6892029107685065
```

While these are somewhat high scores, it is not close to the performance of the other algorithms. Therefore, it is clear that this data is not easily linearly separable and is non-trivial.

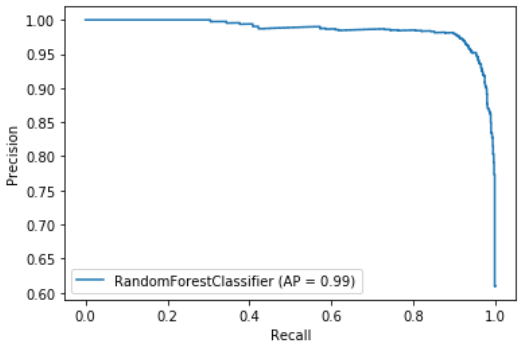
Comparing Algorithm Performance

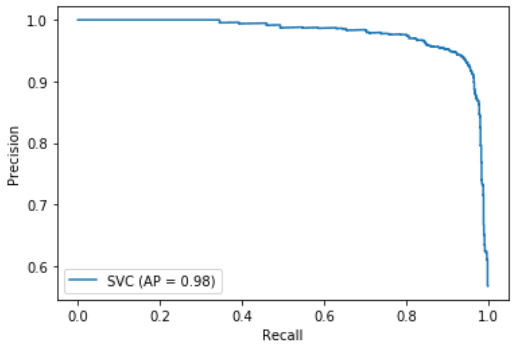
In order to compare the algorithms, their performance with the optimal hyperparameters on an unseen testing dataset was recorded. In order to test multiple instances, a KFold was used. The results of each of the algorithms are as follows.

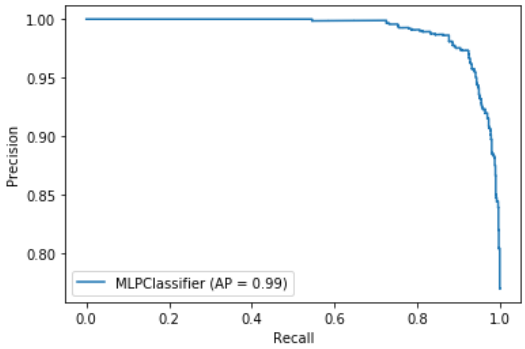
```
Random Forest Mean 0.9678814561167502
Random Forest SD 0.024989996315341143
SVM Mean 0.9525070319187966
SVM SD 0.019123903273824107
Neural Network Mean 0.9619991031755738
Neural Network SD 0.02115397201514253
```




```
Blue = Random Forest
Red = SVM
Green = Neural Network
```







From the performance metrics, it is clear that all of the algorithms performed quite well. They all had an accuracy of over 95%. For sheer performance though, the random forest performed the best as it had the highest mean of accuracy. However, the support vector machine had the lowest standard deviation, giving the most consistent results. For all of the 10 different trials, they all had a very similar accuracy as shown in the graph. In addition, they all had a very large area under the curve for the precision-recall graph. This means that they have a low false positive rate along with a low false negative rate. While similar, it appears that the random forest has the largest area, followed by the support vector machine, and then the neural network. While each of the algorithms do a great job at modeling this data with accuracy, the random forest is slightly more accurate than the rest, making it the best performance pick.

Conclusion

Other than performance, the best algorithm is most likely still the random forest. While it did take longer to determine the optimal hyperparameters than the support vector machine, it was significantly faster to hyper tune than the neural network. The neural network took far longer to hyper tune than the others and also had many more options for the hyperparameters, making it even less effective.

When actually training on real data after the hyperparameters were set, the random forest and support vector machine took about the same amount of time. The neural network took significantly longer to train and therefore would be less practical to use in a real-world situation. Although, in the real world like this would most likely be done on a much more powerful server, making this time difference somewhat negligible. Since it has the best performance, best training time, and easiest hyperparameter tuning, the random forest algorithm is clearly the best for this dataset.

Acknowledgements

Scikit-learn was used to generate all of the classifiers along with the metrics regarding those classifiers.