# Term Project – Final Report

SYSC5405 - 12/18/2019

Group 7 – Decision Trees

Prepared for Dr. James Green

Prepared by Jason Miller, Ben Earle,
Mohamed Hozayen, and Ian Showalter

# 1. Introduction

Group 7 chose to use a decision tree for their regular classifier and Adaptive Boosting (AdaBoost) for their meta learning algorithm based on the assumption that the project would present a heavy class imbalance. Decision trees and their meta learning variants are common solutions for class imbalance problems [1]. The code written for this project is available in [2] and was all done in python using many popular machine learning libraries: SciKit-Learn, Pandas, Seaborn, Imbalanced-Learn, MatPlotLib, Scipy and Numpy.

# 2. Experimental Design

The primary goal for the designed experiment was to achieve a good prediction of our Precision at Recall greater than 50% (Pr@Re>50). The experiment's flow can be seen in the following figure.

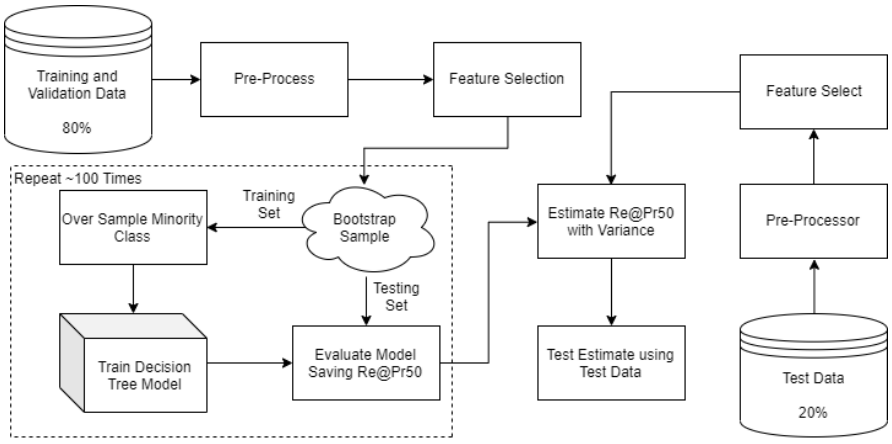


*Figure 1: Experimental Design*

The data was partitioned to use 80% for Training and Validation and the remaining 20% as a hold out test. The larger data partition was used to optimize the pre-processing, feature selection and hyper parameters, before performing bootstrap testing, and finally training the final model. The hold out test was then used as a sanity check for the bootstrapping test statistics and help identify any overfitting. The model retained for our final submission was only trained on the Training and Validation set to ensure the hold out test would emulate the final blind test.

# 3. Preprocessing

Outliers were detected using 1.5 inter-quartile (IQ) rule. However, the 1.5 IQ rule was applied around the median since the mean was significantly affected by extreme outliers. *Figure 2: Data status and corrupt mean* shows the data status before processing. Looking at the Z1_NO_sideR35_CV feature, 75% of the data are between -2.07 and 8.05, but extreme outliers are clearly shown in the maximum and minimum values. Moreover, when comparing the mean with the median, the mean is significantly affected by outliers. **Error! Reference source not found.**

| Index | IO_sideL35_M | HP_NO_sideL35_CV | Z1_NO_sideR35_CV | b_NO_sideR35_S | ›_NC |
|---|---|---|---|---|---|
| count | 38 | 19988 | 19988 | 19988 | 1998 |
| mean | 3515 | -1.54499e+12 | -1.37108e+13 | 0.824004 | 0.54 |
| std | 0637 | 4.10079e+15 | 2.2473e+15 | 0.55734 | 0.07 |
| min | 51.96 | -1.83949e+17 | -2.72061e+17 | -1.83 | 0.04 |
| 25% | 6 | -7.4 | -2.07 | 0.59 | 0.52 |
| 50% | 1 | -3.65 | 3.74 | 0.81 | 0.55 |
| 75% | 4 | -1.54 | 8.0525 | 1.04 | 0.59 |
| max | 7.05 | 2.80225e+17 | 1.21199e+17 | 5.36 | 0.7 |

Figure *2*: Data status and corrupt mean

The 1.5 IQ rule around median was applied iteratively until the ratio of detected outliers between two consecutive iterations became insignificant (ratio=1.07). *Figure 3: Outlier detection using iterative 1.5 IQ rule around median* shows the effectiveness of using 1.5 IQ rule iteratively for detecting outliers.

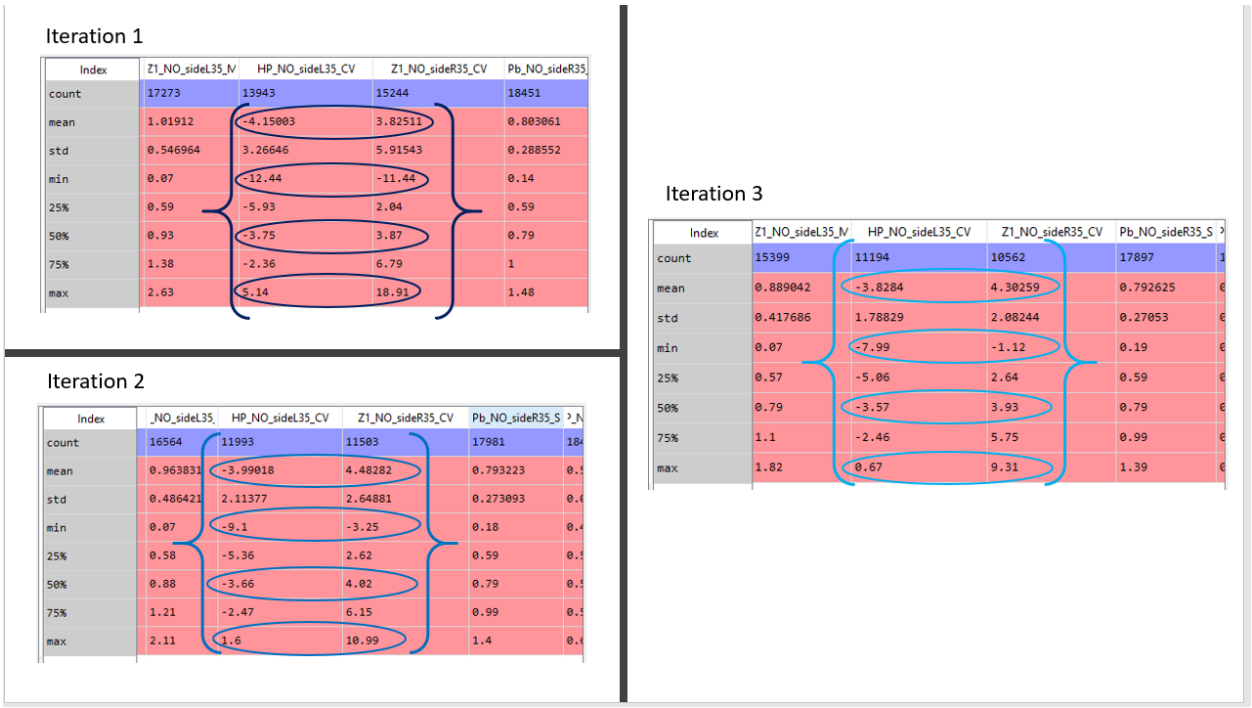


Figure *3*: Outlier detection using iterative 1.5 IQ rule around median

After detecting outliers, samples with more than 10 outliers out of 29 features were identified as weak samples and removed entirely. Then, all outliers were replaced with the median prior to the data being standardized.

## 4. Optimal Feature Selection

Optimal features give the best precision at recall 50% with the optimal tree depth with respect to overfitting precautions. *Figure* 4*: Optimal feature selection design* illustrates an exhaustive methodology for optimal feature selection. Unsupervised feature selection methods were applied to processed data with the evaluation of precession at recall 50% at different tree depth for different methods.

Principle complement analysis (PCA) with rbf, poly, and cosine kernels dominated all other methods with tree depth of four and a 7-8% precession at recall 50%.

Afterwards, the data were balanced using weight-balancing and supervised methods were applied similarly to unsupervised methods, but on the new PCA dimensions.

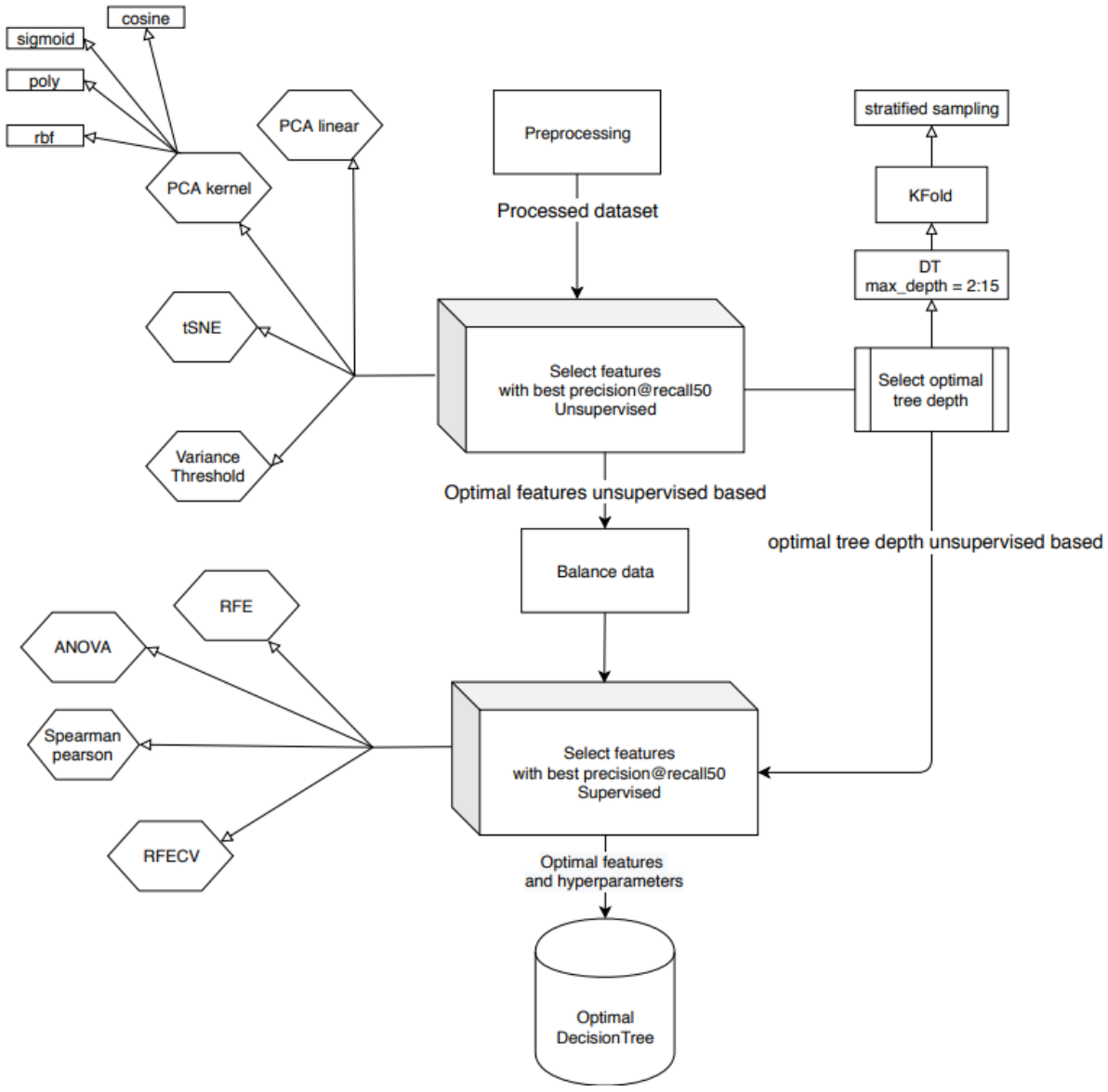


Figure 4: Optimal feature selection design

According to *Figure* 5*: The optimal feature with optimal depth*, the optimal features are obtained using PCA with a cosine kernel at tree depth of four with precision of 7.8% at recall 50% after using Recursive Feature Elimination with Cross Validation (RFECV) selecting best 5 features of PCA-Cosine; feature 12, 13, 15, 18, 26.

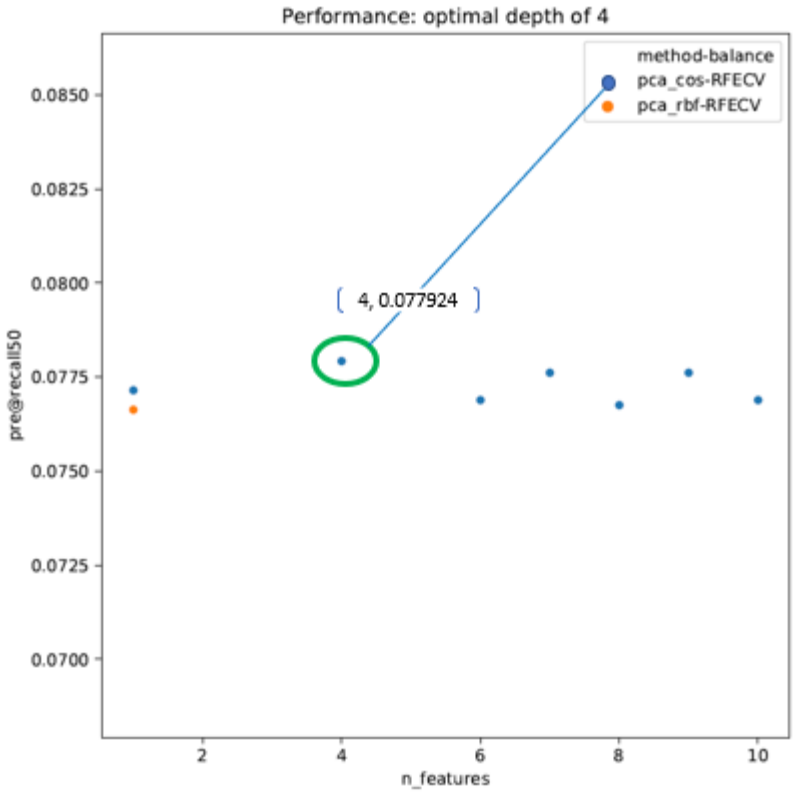


Figure *5*: The optimal feature with optimal depth

**The beauty of engineering:** Data preprocessing and optimal feature selection are fully automated using Group 7 custom-made libraries with only 9 lines of codes! See *Figure* 6 *summary of preprocessing and optimal feature selection*

```
import pandas as pd
import preprocessing as prc
import feature_selection as fs

df = pd.read_csv('Files\csv_result-Descriptors_Training.csv', sep=',')
df = df.drop(['id'], axis=1)
df = prc.handle_outlier(prc.detect_outlier_iterative_IQR(df))
df = prc.standarize(df)

pca_cos_features = fs.pca_kernel(df, kernel='cosine')
features = pca_cos_features[['pca-cosine13', 'pca-cosine18', 'pca-cosine15', 'pca-cosine12', 'pca-cosine26']]

"""
DecisionTree_depth-4 predict (features)
"""
```

Figure *6* summary of preprocessing and optimal feature selection

# 5. Training

Group 7 used a decision tree as a regular classifier and AdaBoost as the meta learning algorithm. This section discusses the approach to training the models, and lessons learned.

## 5.1. Decision Tree

Decision trees are known to be brittle classifiers; therefore, it is important to perform regularization for them to generalize. Additionally, without limiting the tree it continues to split until the leaf nodes are pure. The predicated probability for a leaf is calculated based on how many of each class falls in this leaf during training. If all the leaf nodes are pure, then the predicted probability of every sample classified will be 100% preventing us from having a good precision recall curve. SciKit-Learn provides several ways of regularizing the decision trees, such as limiting the depth of the tree, the number of leaves, the samples to split, the samples per leaf, or the minimum impurity gain per split. These regularization methods were all tested, and it became apparent that limiting the depth of the tree was the simplest method in addition to performing equally well. Pruning was not used since the data set was quite large and it was thought to be too computationally expensive. Retrospectively, we should have experimented with this further. The next step was to find the optimal tree depth given experimental design constraints. A test was created to experimentally determine the best depth given a data set; it used five-fold stratified cross validation on the Training and Validation set and output the Pr@Re>50. Contrary to our expectations, performing preprocessing, feature selection, and balancing of the data decreased our Pr@Re>50. Upon further investigation it was found that, while the test statistic worsened, our classifier allowed for better generalization. For example, when evaluated against an unbalanced data set, the optimal tree depth was 8. In contrast, balancing the data using weighted errors resulted in an optimal depth of four. Since the decision tree grows exponentially with depth, reducing the depth by four significantly decreases the probability of overfitting. The following figure shows the test statistic for various tree depths.

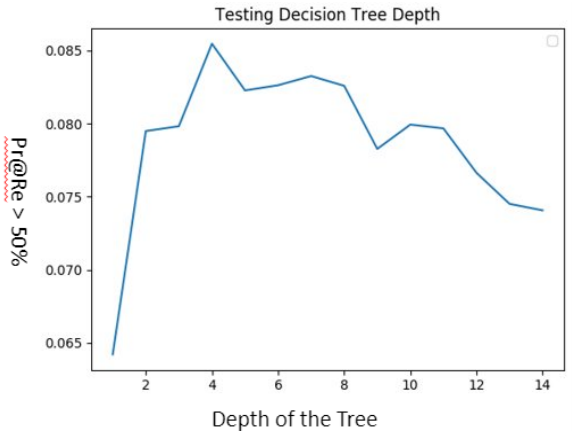


Figure 7: Test Statistic Compared to Tree Depth

## 5.2. Adaptive Boosted Decision Trees

Since AdaBoost fits extremely well with a decision tree classifier, the main hyperparameter to be determined was the number of decision trees to boost, as too many decision trees would lead to overfitting of the data. Given that it had been determined that a depth of four was the optimal decision tree depth, it followed that the AdaBoost meta-learning algorithm should be applied to the same style of tree.

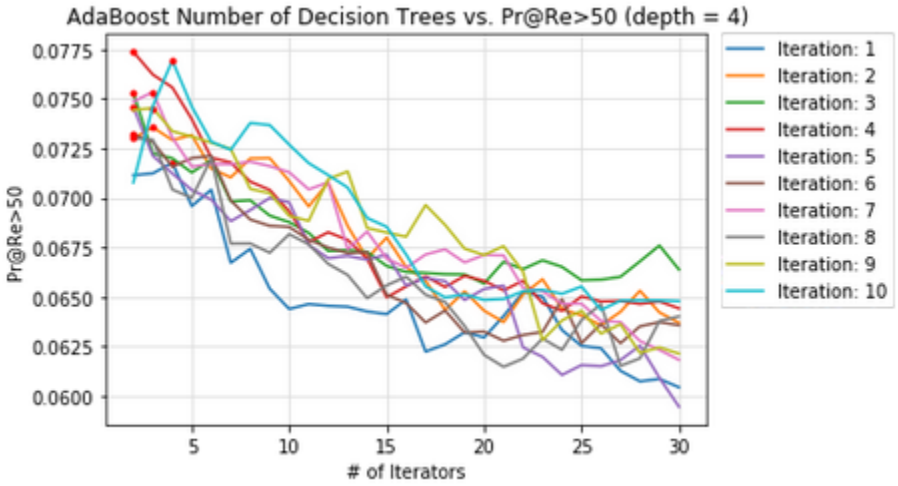


*Figure 8: Number of AdaBoost Decision Trees with depth 4 vs Precision at Recall of at least 50%*

As can be seen in Figure 8, the optimal number of AdaBoosted decision trees would be extremely low (indicated by the red dots between one and four decision trees), as more decision trees would cause the "Precision at Recall > 50%" to drop at an alarming rate. In retrospect, this makes sense as the data

would have been optimally fitted for one decision tree of depth four and not gain much benefit from having more trees of the same depth.

Further investigation showed that a depth of one produced extremely good results. This was reinforced by further research into the underlying theory behind AdaBoost. Since AdaBoost works best on weak learners (i.e.: a decision tree with only one depth, called a decision stump), it is extremely advantageous (and simple) to limit the decision tree such that it will be a weak learner. Once the decision trees were limited to a depth of one, the same test was rerun to determine if an optimal number of decision trees could be found.

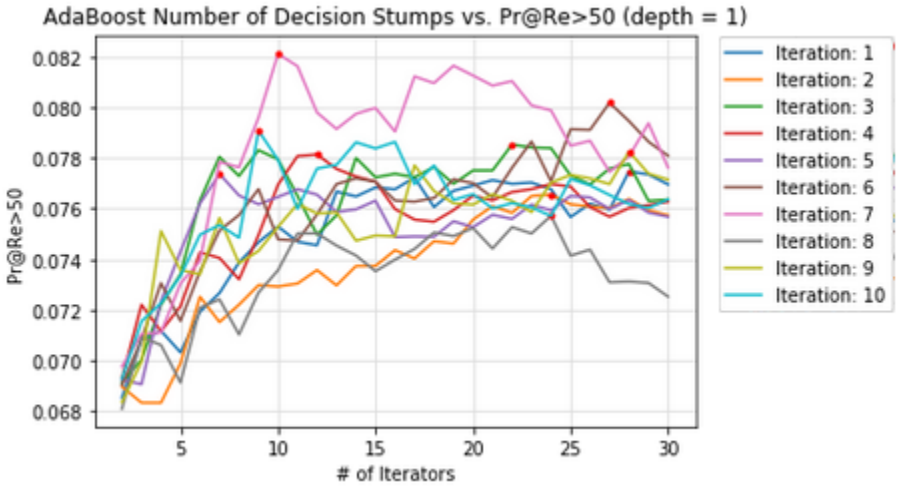


*Figure 9: Number of AdaBoost Decision Stumps with Depth 1 vs Precision at Recall of at Least 50%*

Figure 9 shows that depending on the original split of the data, there is not only a large amount of variance between the number of optimal decision stumps (ranging from about 7 to 28), but there is also variance in the "Pr@Re>50" between the highest and lowest performer. It was decided that 22 decision stumps would be used, as the variance looked to be manageably small (this was done by inspection, looking at Figure 9). While ten decision stumps provided the single best PR@Re>50 value, the difference between the highest and the lowest precisions was decided to be too wide to give a confident prediction.

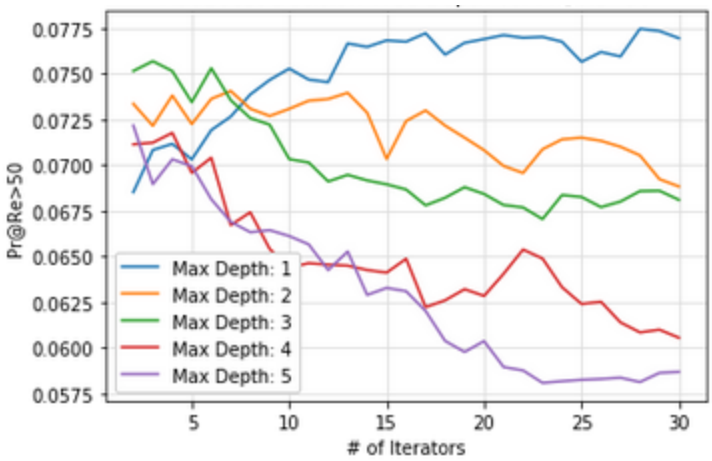


*Figure 10: Number of AdaBoosted Decision Trees ranging from depth 1 to depth 5*

In order to show that 22 decision stumps would be a valid collection of AdaBoosted stumps, a comparison of depth 1 to 5, inclusive, is presented in Figure 10. It should be noted that this figure was generated over one sample of five-fold cross validation and is used to show the trend of using varying depths. It is expected that these results should hold for any arbitrary splitting of the data (i.e.: 22 AdaBoosted decision trees of depth 1 will outperform 22 AdaBoosted decision trees of any other depth).

# 6. Testing

Four tests were applied to both the trained decision tree and AdaBoost decision tree models. A holdout test was used to estimate the true error. A bootstrap test was used to determine statistically relevant performance metrics. A permutation test was used to compare each model to a random classifier, and a brief literature review was used to determine state-of-the-art performance metrics.

## 6.1. Holdout Test

A 20% subset of the combined datasets obtained from the `csv_result-Descriptors_Training.csv', and `csv_result-Descriptors_Calibration.csv' was reserved for holdout testing. The trained models were then applied to the holdout test data.

### 6.2. Bootstrap Test

A bootstrapping test of 100 iterations was applied to the 80% of the data not withheld for the holdout test. The models were each retrained 100 times on a 50% stratified data split with replacement. The precision at recall greater than 50% was calculated at each iteration. At completion of the bootstrap test, the mean and standard deviation of the precision value were calculated.

### 6.3. Permutation Test

For each model a 100-times accuracy-based permutation test was performed on the test data subset. This was then compared to a random classifier's accuracy of 50%.

### 6.4. State-of-the-Art

A brief literature search retrieved no documents where precision at recall greater than 50%, so accuracy was used to compare the two models presented here with state-of-the-art results from peer-reviewed publications.

## 7. Results

### 7.1. Holdout and Bootstrap Test Results

The results of the bootstrapping and holdout tests for the decision tree and AdaBoost decision tree are presented in Table I, and Figure 11.

| **Maximum Precision at Recall>50%** | **Decision Tree** | **AdaBoost Decision Tree** |
|---|---|---|
| Apparent Error | 0.0769 | 0.0868 |
| 0.632 Bootstrap Error | 0.0705 | 0.0763 |
| Bootstrapping (mean, std) | 0.0668, +/- 0.00406 | 0.0701, +/- 0.00316 |
| Holdout test | 0.0694 | 0.0700 |
| Accuracy | 0.496 | 0.604 |

*Table I: Precision Results*

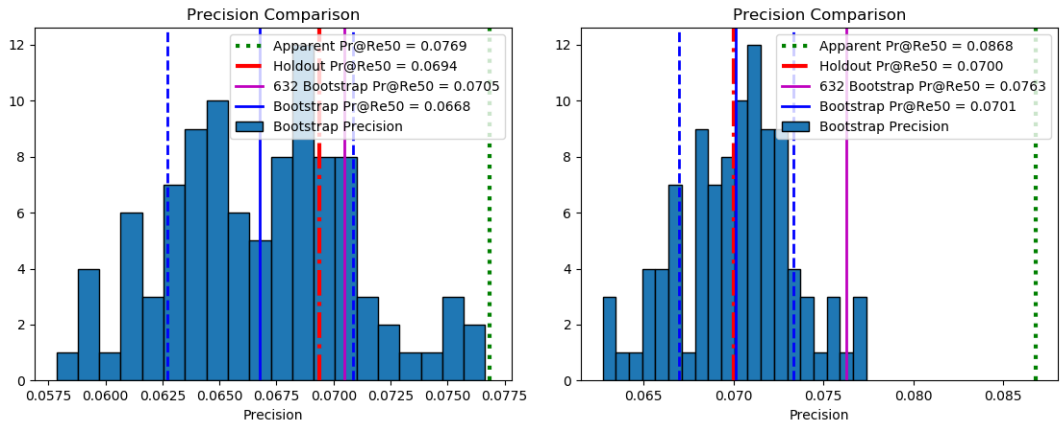


*Figure 11: Precision Results, Left: Decision Tree, Right: AdaBoost Decision Tree*

## 7.2. Permutation and State-of-the-Art Test Results

The results of the permutation tests for the decision tree and AdaBoost decision tree are presented in Figure 12. The decision tree accuracy was very close to chance The AdaBoost decision tree accuracy was slightly better than chance at 0.604, with a p-value of 0.0297. State-of-the-art accuracy would more likely be in the range of 76.2%-80.7% based on the results presented in [3].

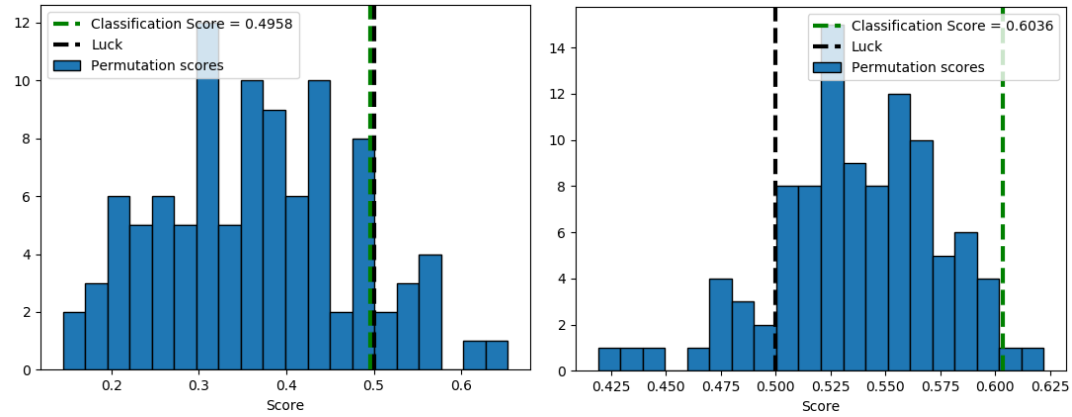


*Figure 12: Permutation Results, Left: Decision Tree, Right: AdaBoost Decision Tree*

# 8. Conclusions

The random number generator seed was set at the beginning of the Python script to produce repeatable results when it was run multiple times. For the AdaBoost decision tree, different initial seeds were tried.

A value of 30 was chosen because the holdout test precision was very similar to the bootstrap mean precision value.

The AdaBoost decision tree predicted precision was chosen slightly optimistically (0.071) based on the bootstrap test mean (0.07). The standard deviation was chosen pessimistically based on the difference between the bootstrap mean, and the 0.632 bootstrap estimate. A comparison of the predicted results and those obtained during the evaluation portion of the project is presented in Table II. The difference between the actual Pr@Re50 and the predicted Pr@Re50 is 0.0036, which is 0.6 of one standard deviation of the predicted Pr@Re50.

*Table II: Actual and Predicted Score*

| Predicted Score | Actual Score 1 | Actual Score 2 |
|---|---|---|
| 0.071, +/-0.006 | 0.0674 | 55.377 |

In retrospect, the predicted precision should not have been chosen slightly optimistically and the standard deviation predicted conservatively. If the bootstrapping mean, or holdout error had been used as the predicted Pr@Re50, the actual Pr@Re50 would have been within 0.85 of the bootstrapping standard deviation (0.00316). This would have significantly increased the team's lead over the second ranked team in Actual Score 2.

# 9. References


[1] D. A. Cieslak and N. V. Chawla, "Learning Decision Trees for Unbalanced Data," *Machine Learning and Knowledge Discovery in Databases Lecture Notes in Computer Science*, pp. 241–256.

[2] M. Hozayen, J. Miller, I. Showalter, and B. Earle, "SYSC5405-Prject" on GitHub. https://github.com/BenEarle/SYSC5405-Project/graphs/contributors.

[3] L. Wei, P. Xing, G. Shi, Z. Ji, and Q. Zou, "Fast Prediction of Protein Methylation Sites Using a Sequence-Based Feature Selection Technique," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 16, no. 4, pp. 1264–1273, Jan. 2019.