IST 707 Final Project

Matt Ryder

3/26/2021

## The Project

The goal of this project is to utilize the classification tools learned in this class to develop algorithms which can predict whether someone has a high chance of heart disease or not.

## The Data

The dataset was obtain from Kaggle and can be found at: <https://www.kaggle.com/nareshbhat/health-care-data-set-on-heart-attack-possibility>

The dataset contains the following attributes: 1) age 2) sex 3) chest pain type (4 values) 4) resting blood pressure 5) serum cholestoral in mg/dl 6)fasting blood sugar > 120 mg/dl 7) resting electrocardiographic results (values 0,1,2) 8) maximum heart rate achieved 9) exercise induced angina 10) oldpeak = ST depression induced by exercise relative to rest 11)the slope of the peak exercise ST segment 12) number of major vessels (0-3) colored by flourosopy 13) thal: 0 = normal; 1 = fixed defect; 2 = reversable defect 14) target: 0= less chance of heart attack 1= more chance of heart attack

heart = read.csv("D:/Winter 2021/IST 707- Data Analytics/Final/heart.csv")

## Data Preprocessing

After reading the data in to R I will split it into a training set and testing set, using random sampling. I will also make the “target” variable a factor so that our alogrithms perform classification problems rather than regression.

set.seed(1)  
  
heart$target = as.factor(heart$target)  
  
train\_size = round(nrow(heart)\*.8,0)  
test\_size = nrow(heart) - train\_size  
  
train\_index = sample(1:nrow(heart), train\_size)  
  
train\_data = heart[train\_index,]  
test\_data = heart[-train\_index,]

## Algorithms

First, I will use three different algorithms which we have studied in class. The first algorithm will be the Random Forest, then k-Nearest Neighbors, and lastly Support Vector Machine. In addition to those algorithms, I will also utilize the eXtreme Gradient Boosting method within the caret package and compare the results.

#### Random Forest

Train a random forest with 3 different tree value to find an optimal model

set.seed(1)  
  
rf\_500 = randomForest(target~., data = train\_data, ntree = 500)  
  
rf\_750 = randomForest(target~., data = train\_data, ntree = 750)  
  
rf\_1000 = randomForest(target~., data = train\_data, ntree = 1000)

Test the models

rf500\_pred = predict(rf\_500, test\_data)  
confusionMatrix(rf500\_pred, test\_data$target, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 2  
## 1 4 34  
##   
## Accuracy : 0.9016   
## 95% CI : (0.7981, 0.963)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 7.834e-08   
##   
## Kappa : 0.7942   
##   
## Mcnemar's Test P-Value : 0.6831   
##   
## Sensitivity : 0.9444   
## Specificity : 0.8400   
## Pos Pred Value : 0.8947   
## Neg Pred Value : 0.9130   
## Prevalence : 0.5902   
## Detection Rate : 0.5574   
## Detection Prevalence : 0.6230   
## Balanced Accuracy : 0.8922   
##   
## 'Positive' Class : 1   
##

rf750\_pred = predict(rf\_750, test\_data)  
confusionMatrix(rf750\_pred, test\_data$target, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 1  
## 1 4 35  
##   
## Accuracy : 0.918   
## 95% CI : (0.819, 0.9728)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 1.172e-08   
##   
## Kappa : 0.8274   
##   
## Mcnemar's Test P-Value : 0.3711   
##   
## Sensitivity : 0.9722   
## Specificity : 0.8400   
## Pos Pred Value : 0.8974   
## Neg Pred Value : 0.9545   
## Prevalence : 0.5902   
## Detection Rate : 0.5738   
## Detection Prevalence : 0.6393   
## Balanced Accuracy : 0.9061   
##   
## 'Positive' Class : 1   
##

rf1000\_pred = predict(rf\_1000, test\_data)  
confusionMatrix(rf1000\_pred, test\_data$target, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 1  
## 1 4 35  
##   
## Accuracy : 0.918   
## 95% CI : (0.819, 0.9728)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 1.172e-08   
##   
## Kappa : 0.8274   
##   
## Mcnemar's Test P-Value : 0.3711   
##   
## Sensitivity : 0.9722   
## Specificity : 0.8400   
## Pos Pred Value : 0.8974   
## Neg Pred Value : 0.9545   
## Prevalence : 0.5902   
## Detection Rate : 0.5738   
## Detection Prevalence : 0.6393   
## Balanced Accuracy : 0.9061   
##   
## 'Positive' Class : 1   
##

After reviewing the confusion matrices, both the 1000 tree and 750 tree model performed identically with a 91.8% accuracy

#### kNN

With a test data set of 61 obs I will build the models with only 3 possible k values of 3, 5, and 7

set.seed(1)  
  
knn7 = knn(train\_data[,-14], test\_data[,-14], train\_data$target, k = 7, prob = FALSE)  
  
knn5 = knn(train\_data[,-14], test\_data[,-14], train\_data$target, k = 5, prob = FALSE)  
  
kNN3 = knn(train\_data[,-14], test\_data[,-14], train\_data$target, k = 3, prob = FALSE)

Test the models

confusionMatrix(knn7, test\_data[,14], positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 15 5  
## 1 10 31  
##   
## Accuracy : 0.7541   
## 95% CI : (0.6271, 0.8554)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 0.005604   
##   
## Kappa : 0.4756   
##   
## Mcnemar's Test P-Value : 0.301700   
##   
## Sensitivity : 0.8611   
## Specificity : 0.6000   
## Pos Pred Value : 0.7561   
## Neg Pred Value : 0.7500   
## Prevalence : 0.5902   
## Detection Rate : 0.5082   
## Detection Prevalence : 0.6721   
## Balanced Accuracy : 0.7306   
##   
## 'Positive' Class : 1   
##

confusionMatrix(knn5, test\_data[,14], positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 13 9  
## 1 12 27  
##   
## Accuracy : 0.6557   
## 95% CI : (0.5231, 0.7727)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 0.1815   
##   
## Kappa : 0.275   
##   
## Mcnemar's Test P-Value : 0.6625   
##   
## Sensitivity : 0.7500   
## Specificity : 0.5200   
## Pos Pred Value : 0.6923   
## Neg Pred Value : 0.5909   
## Prevalence : 0.5902   
## Detection Rate : 0.4426   
## Detection Prevalence : 0.6393   
## Balanced Accuracy : 0.6350   
##   
## 'Positive' Class : 1   
##

confusionMatrix(kNN3, test\_data[,14], positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 13 7  
## 1 12 29  
##   
## Accuracy : 0.6885   
## 95% CI : (0.5571, 0.801)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 0.07456   
##   
## Kappa : 0.3358   
##   
## Mcnemar's Test P-Value : 0.35880   
##   
## Sensitivity : 0.8056   
## Specificity : 0.5200   
## Pos Pred Value : 0.7073   
## Neg Pred Value : 0.6500   
## Prevalence : 0.5902   
## Detection Rate : 0.4754   
## Detection Prevalence : 0.6721   
## Balanced Accuracy : 0.6628   
##   
## 'Positive' Class : 1   
##

The best performing model was the 7-neighbor model with an accuracy of 75.41%, due to the high false positive rate and significantly lower accuracy than the random forest models.

#### Support Vector Machine

set.seed(1)  
  
svm\_linear = svm(target~., data = train\_data, scale = FALSE, kernel = "linear", type = "C")  
  
svm\_poly = svm(target~., data = train\_data, scale = FALSE, kernel = "polynomial", type = "C")  
  
svm\_radial = svm(target~., data = train\_data, scale = FALSE, kernel = "radial", type = "C")  
  
svm\_sigmoid = svm(target~., data = train\_data, scale = FALSE, kernel = "sigmoid", type = "C")

Test the different kernels

svm\_linear\_pred = predict(svm\_linear, test\_data)  
confusionMatrix(rf500\_pred, test\_data$target, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 2  
## 1 4 34  
##   
## Accuracy : 0.9016   
## 95% CI : (0.7981, 0.963)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 7.834e-08   
##   
## Kappa : 0.7942   
##   
## Mcnemar's Test P-Value : 0.6831   
##   
## Sensitivity : 0.9444   
## Specificity : 0.8400   
## Pos Pred Value : 0.8947   
## Neg Pred Value : 0.9130   
## Prevalence : 0.5902   
## Detection Rate : 0.5574   
## Detection Prevalence : 0.6230   
## Balanced Accuracy : 0.8922   
##   
## 'Positive' Class : 1   
##

svm\_poly\_pred = predict(svm\_poly, test\_data)  
confusionMatrix(rf750\_pred, test\_data$target, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 1  
## 1 4 35  
##   
## Accuracy : 0.918   
## 95% CI : (0.819, 0.9728)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 1.172e-08   
##   
## Kappa : 0.8274   
##   
## Mcnemar's Test P-Value : 0.3711   
##   
## Sensitivity : 0.9722   
## Specificity : 0.8400   
## Pos Pred Value : 0.8974   
## Neg Pred Value : 0.9545   
## Prevalence : 0.5902   
## Detection Rate : 0.5738   
## Detection Prevalence : 0.6393   
## Balanced Accuracy : 0.9061   
##   
## 'Positive' Class : 1   
##

svm\_radial\_pred = predict(svm\_radial, test\_data)  
confusionMatrix(rf1000\_pred, test\_data$target, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 1  
## 1 4 35  
##   
## Accuracy : 0.918   
## 95% CI : (0.819, 0.9728)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 1.172e-08   
##   
## Kappa : 0.8274   
##   
## Mcnemar's Test P-Value : 0.3711   
##   
## Sensitivity : 0.9722   
## Specificity : 0.8400   
## Pos Pred Value : 0.8974   
## Neg Pred Value : 0.9545   
## Prevalence : 0.5902   
## Detection Rate : 0.5738   
## Detection Prevalence : 0.6393   
## Balanced Accuracy : 0.9061   
##   
## 'Positive' Class : 1   
##

svm\_sigmoid\_pred = predict(svm\_sigmoid, test\_data)  
confusionMatrix(rf1000\_pred, test\_data$target, positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 1  
## 1 4 35  
##   
## Accuracy : 0.918   
## 95% CI : (0.819, 0.9728)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 1.172e-08   
##   
## Kappa : 0.8274   
##   
## Mcnemar's Test P-Value : 0.3711   
##   
## Sensitivity : 0.9722   
## Specificity : 0.8400   
## Pos Pred Value : 0.8974   
## Neg Pred Value : 0.9545   
## Prevalence : 0.5902   
## Detection Rate : 0.5738   
## Detection Prevalence : 0.6393   
## Balanced Accuracy : 0.9061   
##   
## 'Positive' Class : 1   
##

All models except the model with the linear kernel performed identically to the random forest models, so we currently have 5 models which all at current best performance.

#### eXtreme Gradient Boosting

set.seed(1)  
  
xgboost = train(target~., data = train\_data, method = "xgbDART")

Test the xgboost model

xgb\_predict = predict(xgboost, test\_data)  
confusionMatrix(xgb\_predict, test\_data[,14], positive = '1')

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction 0 1  
## 0 21 2  
## 1 4 34  
##   
## Accuracy : 0.9016   
## 95% CI : (0.7981, 0.963)  
## No Information Rate : 0.5902   
## P-Value [Acc > NIR] : 7.834e-08   
##   
## Kappa : 0.7942   
##   
## Mcnemar's Test P-Value : 0.6831   
##   
## Sensitivity : 0.9444   
## Specificity : 0.8400   
## Pos Pred Value : 0.8947   
## Neg Pred Value : 0.9130   
## Prevalence : 0.5902   
## Detection Rate : 0.5574   
## Detection Prevalence : 0.6230   
## Balanced Accuracy : 0.8922   
##   
## 'Positive' Class : 1   
##

This model had a slightly lower accuracy at 90.16%

#### Summary

After reviewing the different model we have a few different options for what we would choose as the best. First, if we choose the highest accuracy we stil have several option and would have to retrain after acquiring additional data to best decide which model we want to utilize. If I had to put one of the 5 identical models into production based on my data, I would utlize the 750 tree random forest model as it is the least computationally intensive of all the model with a 91.8% accuracy. You could also choose to focus on accurately predicting positive or negative cases. In this example, if one model better predicted people who were high risk of heart disease, at a sacrifice to accuracy, that would be preferred so that we can ensure those who need treatment, get the appropraite treatment. In this case, we do not have a model that better identifies positive cases any more accurately than the models with the highest overall accuracy.

## Packages Used

* randomForest
* e1071
* class
* caret