Binary Classification for Heart Disease

STP 598 Machine Learning

Group members: Ruoqian Liu, Yanyang Tang

4/30/2021

1. Introduction

1.1 Background

Heart disease is the leading cause of death in the United States, causing about 1 in 4 deaths [1]. It is strongly associated with several types of cardiovascular conditions, including diseased vessels, structural problems, and blood clots. The diagnosis of heart disease is performed based on a combination of clinical signs and test results. Depending on the physical exam and medical history, certain types of tests may be performed to diagnose the disease, ranging from electrocardiograms and cardiac computerized tomography (CT) scans, to blood tests and exercise stress tests [2].

We have decided to take a dataset on heart disease from Kaggle (https://www.kaggle.com/ronitf/heart-disease-uci) to study the indications of heart health and predict certain cardiovascular events. This dataset provides a dozen of patients' attributes along with a diagnostic condition of having or not having heart disease [3-6]. Below, the data is first used in a simple logistic model that include all variables, and then the model is investigated using a variety of machine learning tools and techniques, including AIC, BIC, LASSO, Ridge, kNN, random forest, boosting, and neural nets. Results of each method are presented in the following sections, and then discussed in terms of their classifying performance as well as pros and cons.

1.2 Data source

The original database composed of four sub-databases (Budapest, Zurich, Basel, Cleveland) contains 76 attributes [3-6], but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by machine learning researchers to this date, and thus is chosen for this study [6]. The "target" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. In the "processed" dataset, "target" has been recoded to binary classes to be concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0). The names and social security numbers of the patients were recently removed from the database, replaced with dummy values.

1.3 Data description

The dataset used for this study contains 303 observations of 13 predictor variables and 1 response variable, which record the demographics and clinical characteristics of 303 patients. A description of each variable is shown in Table 1.

Table 1. The description of all variables in the dataset.

No.	Attributes	Type	Description
1	age	Numerical	Age in years
2	sex	Categorical	Sex (1 = male; 0 = female)
3	cp	Categorical	Chest pain type (1 = typical angina; 2 = atypical angina; 3 = non-anginal pain; 4 = asymptomatic)
4	trestbps	Numerical	Resting blood pressure (in mm Hg on admission to the hospital)
5	chol	Numerical	Serum cholesterol in mg/dl
6	fbs	Categorical	Fasting blood sugar (> 120 mg/dl; 1 = true; 0 = false)
7	restecg	Categorical	Resting electrocardiographic results (1 = normal; 2 = having ST-T wave abnormality; 2 = showing probable or definite left ventricular hypertrophy by Estes' criteria)
8	thalach	Numerical	Maximum heart rate achieved
9	exang	Categorical	Exercise induced angina (1 = yes; 0 = no)
10	oldpeak	Numerical	ST depression induced by exercise relative to rest
11	slope	Categorical	Slope of the peak exercise ST segment (1 = upsloping; 2 = flat; 3 = downsloping)
12	ca	Categorical	Number of major vessels (0–3) colored by fluoroscopy
13	thal	Categorical	A blood disorder called thalassemia (3 = normal; 6 = fixed defect; 7 = reversible defect)
14	target	Categorical	Diagnosis of heart disease (0 = no presence; 1 = presence)

We first performed the exploratory data analysis to check the distribution of all variables in the dataset. Numerical data were scaled to be suitable for the following data modeling and interpretation. The distribution of numerical data is shown in Figure 1. By inspection, a majority of numerical variables, namely age, trestbps, chol, and thalach, approximately follow a normal distribution, while oldpeak displays a severely right-skewed distribution (Figure 1).

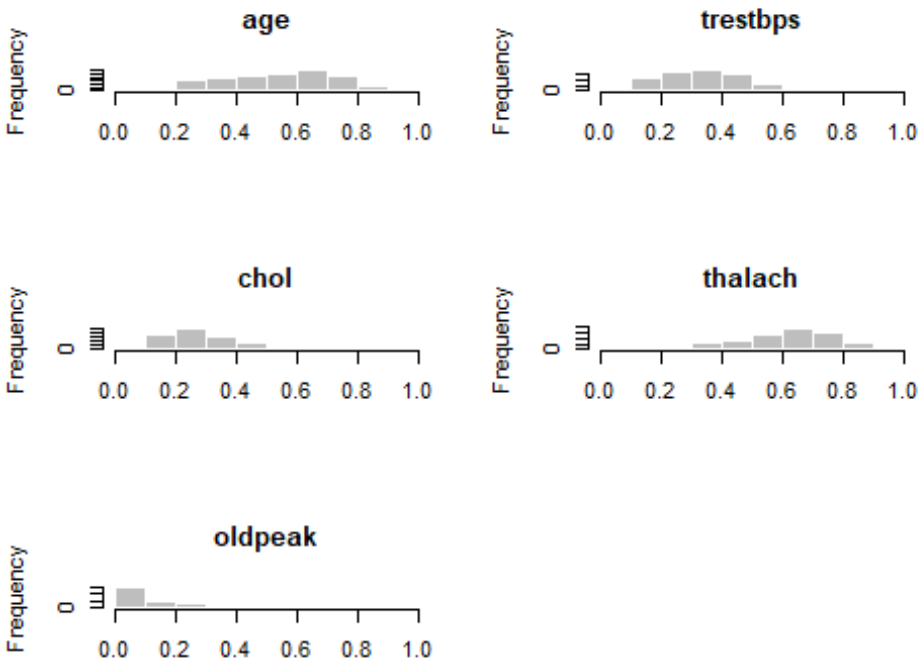


Figure 1. The distribution of numerical variables.

The distribution of categorical data is shown in Figure 2. Some values of categorical variables have very few or no observations, which may not be representative of the true population (Figure 2). One thing to note is that there are more subjects diagnosed with heart disease than those without it (see histogram for "target" in Figure 2). The resulting overrepresentation of patients in the dataset as opposed to the population may artificially promote the classifying performance of models.

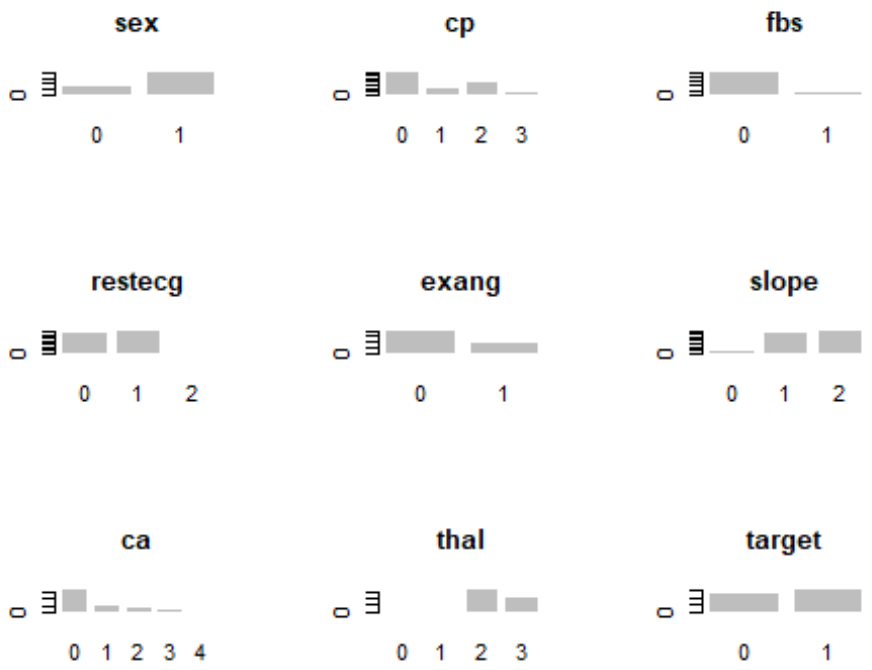


Figure 2. The distribution of categorical variables.

We then performed Pearson's correlation analysis to check the multicollinearity of numerical variables. As shown in Figure 3, the Pearson's correlation coefficients are relatively small, suggesting that there are no strong correlations among numerical variables.

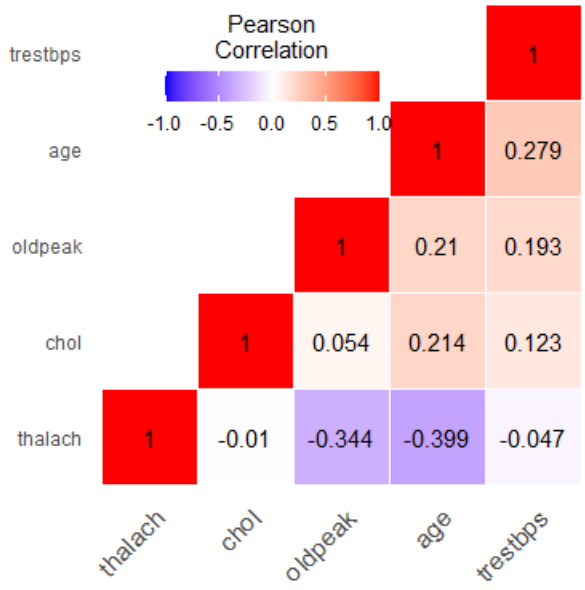


Figure 3. The color-coded Pearson's correlation matrix of numerical variables.

2. Machine Learning Methods

We employed a variety of machine learning tools and techniques to study the binary classification of the condition of heart disease in the dataset in hope of 1) improving the predictive power of a classification model for the heart disease; 2) comparing the strength and weakness of each modeling method. The methods explored in this study include: Logistic regression with four different subset selection methods, kNN, random forest, boosting, and neural nets. Models developed on the train data with the optimal tuning parameter of each method were then fit on the independent test data. Finally, the out-of-sample classifying performance of each model was evaluated by ROC analysis and AUC.

2.1 Logistic regression

Logistic regression is a key model in that it allows us to use ideas from the linear modeling to predict a binary outcome. Here, for logistic regression, we started off with a full model that includes all predictors, and then reduced the model by searching for the proper subset of variables with AIC, BIC, LASSO, or Ridge.

2.1.1 Full model

A full model for logistic regression involving all 13 predictors was generated from the train data and then fitted to the test data. An AUC of 0.870 was achieved by ROC analysis, which is not bad (Figure 4). Logistic regression returns the probability of having such disease. To classify subjects in the two categories: normal group and disease group, we need to find a threshold of probability. Our optimal cutoff was found by maximizing the Youden index, i.e., sensitivity + specificity – 1. Then we obtained four cutoff values that have the exact same maximum Youden index and thus equally good separating ability (Table 2 and Figure 5). The four corresponding pairs of sensitivity and specificity at each cutoff are shown in Table 2.

Table 2. The classification metrics of the full model using optimal Youden cutoffs.

```
##      Cutoffs Sensitivity Specificity    Youden
## 34 0.4028204   0.9210526   0.7894737 0.7105263
## 36 0.4239047   0.8947368   0.8157895 0.7105263
## 38 0.5584640   0.8684211   0.8421053 0.7105263
## 42 0.6406056   0.8157895   0.8947368 0.7105263
```

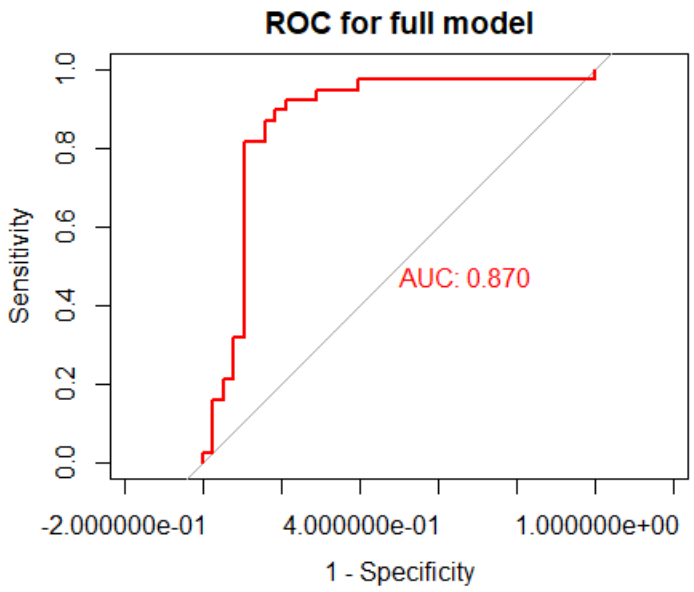


Figure 4. The ROC curve of full model on the test data.

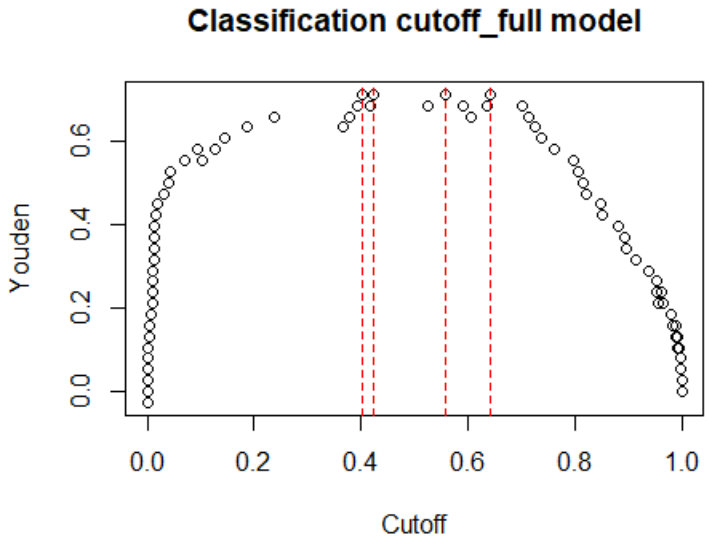


Figure 5. The Youden index at different cutoffs for the full model on the test data.

2.1.2 Subset selection with AIC

The bias-variance tradeoff tells us that the most complicated model may not be the best model. AIC and BIC are widely used attempts to guide us towards simpler models. Both information criteria are composed of two parts: deviance and complexity penalty for number of parameters in the model. When performing these procedures, we used the binomial deviance, i.e., - 2(log likelihood), to examine the fit of our classification models. As we add parameters, the deviance

will go down, but the complexity penalty will go up. So our goal is to find the sweet spot in between, which minimizes AIC or BIC.

We first investigated AIC procedure by examining different number of predictors to be included in the logistic regression model. The lowest AIC of 161.5880 was found when there are 8 predictors in the model (Figure 6), which are sex, cp, trestbps, exang, oldpeak, slope, ca, and thal.

The selected predictors were included for modeling on the train data and the resulting model was fitted to the test data. An AUC of 0.868 was achieved by ROC analysis (Figure 7), which is very close to that of the full model. By maximizing the Youden index, we obtained two cutoff values that have the exact same maximum Youden index and thus equally good separating ability (Table 3 and Figure 8). The two corresponding pairs of sensitivity and specificity at each cutoff are shown in Table 3.

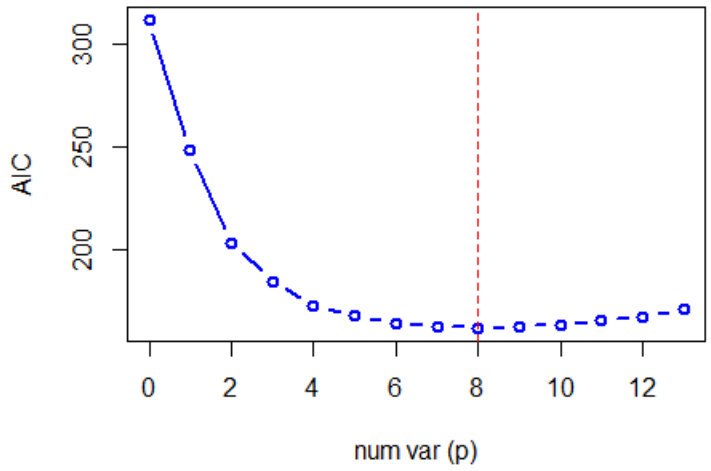


Figure 6. The AIC over different number of predictors in the logistic regression model.

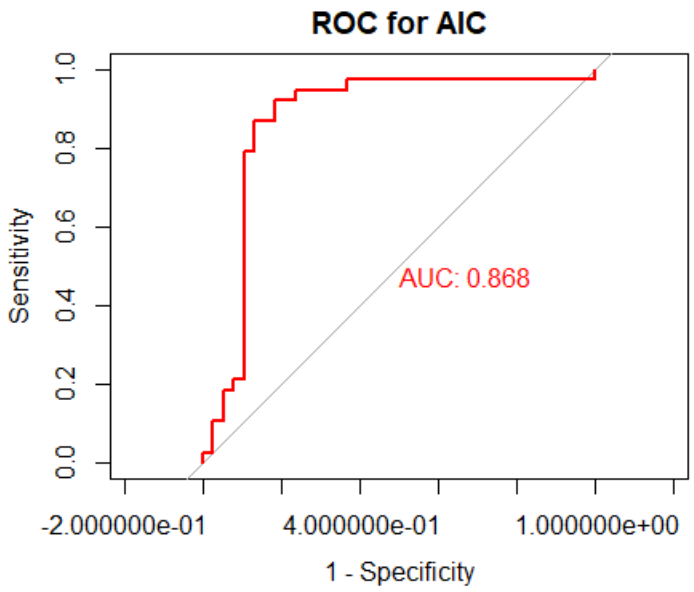


Figure 7. The ROC curve of AIC logistic model on the test data.

Table 3. The classification metrics of the AIC model using optimal Youden cutoffs.

```
##      Cutoffs Sensitivity Specificity    Youden
## 35 0.4287412   0.9210526   0.8157895 0.7368421
## 39 0.5739492   0.8684211   0.8684211 0.7368421
```

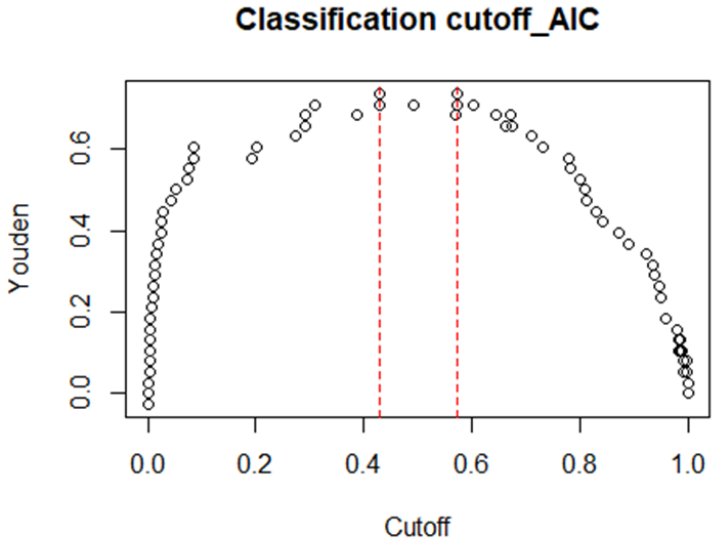


Figure 8. The Youden index at different cutoffs for the AIC model on the test data.

2.1.3 Subset selection with BIC

The BIC is an approximation to the Bayesian approach to model selection which produces an automatic penalty for complex models. Similar to AIC, we investigated BIC procedure also by examining different number of predictors to be included in the logistic regression model. The lowest BIC of 206.7565 was found when there are 6 predictors in the model (Figure 9), which are sex, cp, trestbps, exang, slope, and ca.

The selected predictors were included for modeling on the train data and the resulting model was fitted to the test data. An AUC of 0.862 was achieved by ROC analysis (Figure 10), which is very close to that of the full and AIC models. By maximizing the Youden index, we obtained only one cutoff value (Table 4 and Figure 11). The corresponding pair of sensitivity and specificity at each cutoff is shown in Table 4.

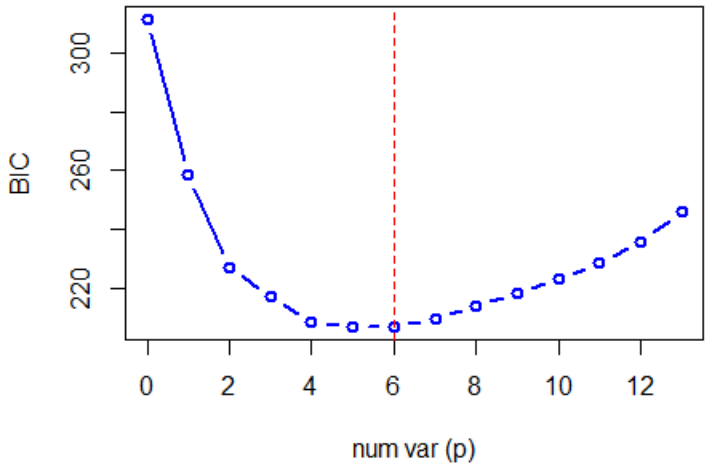


Figure 9. The BIC over different number of predictors in the logistic regression model.

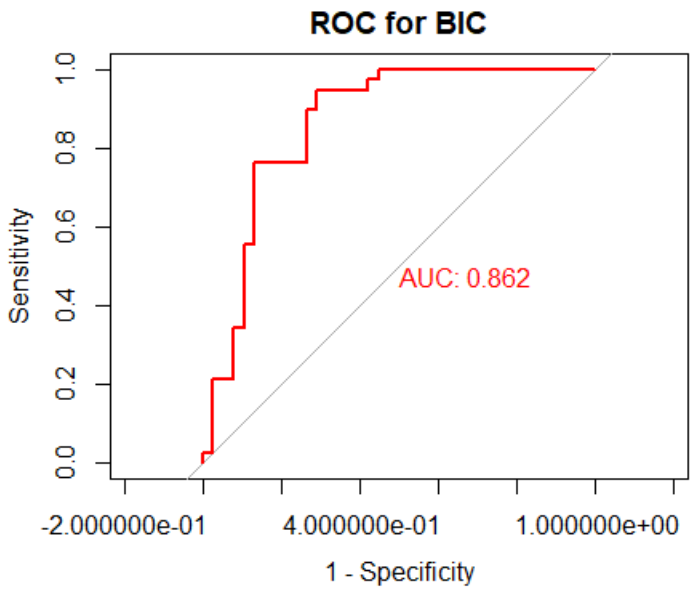


Figure 10. The ROC curve of BIC logistic model on the test data.

Table 4. The classification metrics of the BIC model using optimal Youden cutoffs.

```
##      Cutoffs Sensitivity Specificity    Youden
## 29 0.3355273   0.9473684   0.7105263 0.6578947
```

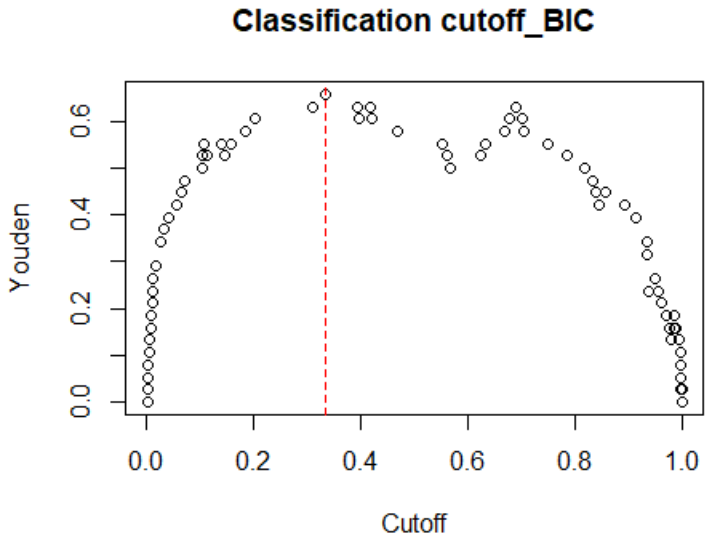


Figure 11. The Youden index at different cutoffs for the BIC model on the test data.

2.1.4 Subset selection with LASSO

In addition to AIC and BIC, regularization is another nice way to explore the bias-variance tradeoff. In this method, a penalty term is added to the deviance, whose weighting parameter λ

controls the extent to which the coefficients are shrunk towards 0. Here, we investigated two regularized logistic regression procedures, i.e., LASSO and Ridge, which are distinguished by their penalty terms.

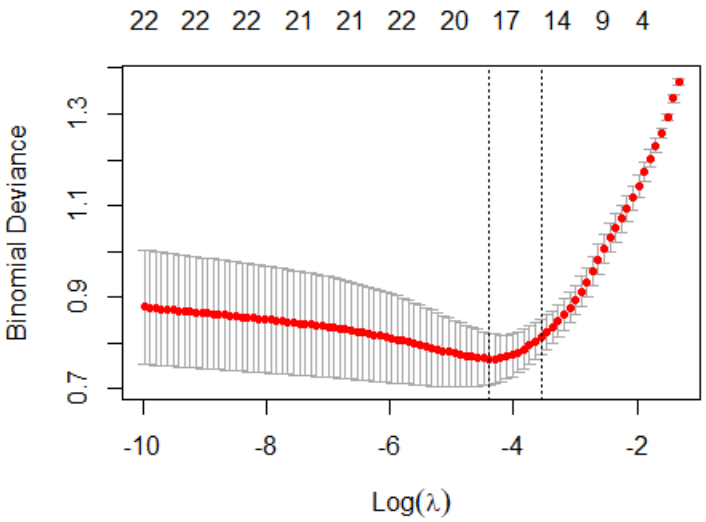


Figure 12. The deviance over different weighting parameter λ of LASSO.

Cross-validation was employed along with LASSO to search for the optimal λ for regularization. As shown in Figure 12, λ min and λ 1se for LASSO are 0.01233446 and 0.0284942, respectively. When the weighting parameter is set to λ 1se, the coefficients of 6 predictors (or dummy variables) are shrunk to 0, including age, chol, fbs1, restecg2, ca4, and thal1 (Figure 13 and Table 5). For the categorical variables that have more than 2 factor levels, we cannot remove those who have the coefficient of only one dummy variable shrunk to 0 from the model. Thus, the selected predictors to be included in the model are sex, cp, trestbps, restecg, thalach, exang, oldpeak, slope, ca, and thal (Table 5).

The LASSO model built on the train data with the selected predictors was then fitted to the test data. An AUC of 0.917 was achieved by ROC analysis (Figure 14), which is notably higher than that of the full, AIC, and BIC models. By maximizing the Youden index, we obtained three cutoff values that have the exact same maximum Youden index and thus equally good separating ability (Table 6 and Figure 15). The three corresponding pairs of sensitivity and specificity at each cutoff are shown in Table 6.

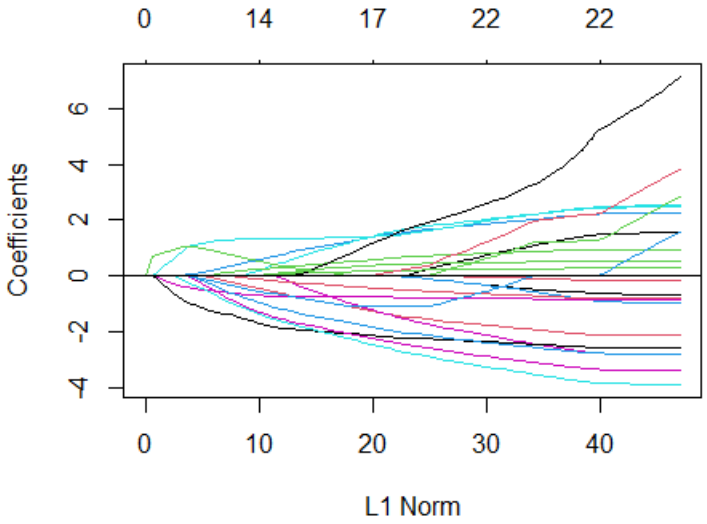


Figure 13. The shrinkage of coefficients for LASSO.

Table 5. The predictor variables selected by LASSO.

```
## (Intercept)  0.77054046
## age           .
## sex1         -0.65539411
## cp1           0.09036520
## cp2           0.77235831
## cp3           0.49059262
## trestbps     -0.11127160
## chol          .
## fbs1          .
## restecg1      0.07361214
## restecg2      .
## thalach       1.32769714
## exang1       -0.72895376
## oldpeak      -1.87643014
## slope1       -0.21171447
## slope2        0.30954953
## ca1          -1.19875872
## ca2          -1.65182578
## ca3          -1.56319639
## ca4           .
## thal1         .
## thal2         0.36979982
## thal3        -0.70843413
```

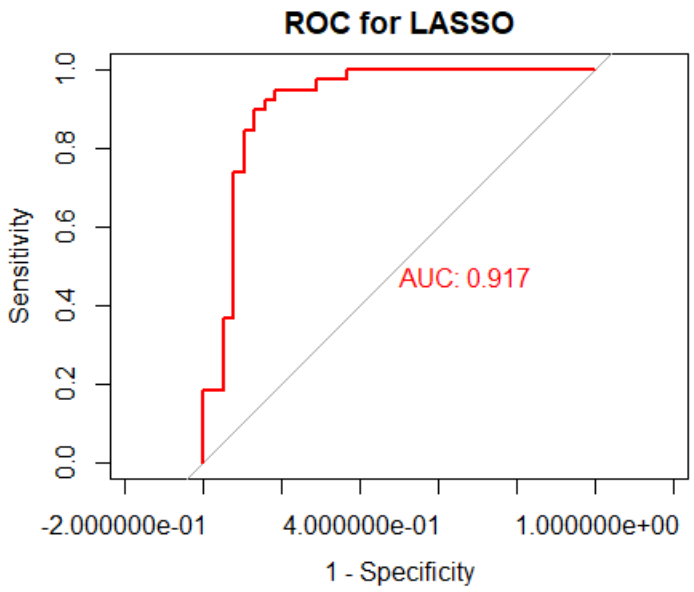


Figure 14. The ROC curve of LASSO logistic model on the test data.

Table 6. The classification metrics of the LASSO model using optimal Youden cutoffs.

```
##      Cutoffs Sensitivity Specificity    Youden
## 34 0.5183977   0.9473684   0.8157895 0.7631579
## 36 0.5409009   0.9210526   0.8421053 0.7631579
## 38 0.5690406   0.8947368   0.8684211 0.7631579
```

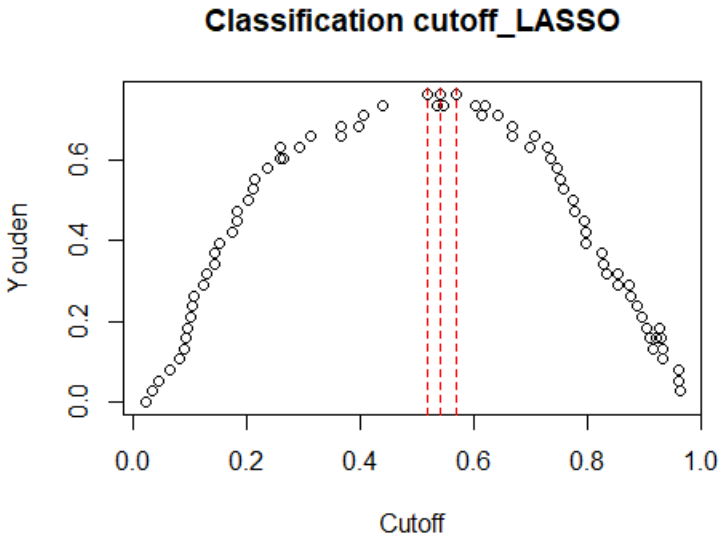


Figure 15. The Youden index at different cutoffs for the LASSO model on the test data.

2.1.5 Subset selection with Ridge

Cross-validation was employed along with Ridge to search for the optimal λ for regularization. As shown in Figure 16, λ min and λ 1se for Ridge are 0.02657379 and 0.1418163, respectively. When the weighting parameter is set to λ 1se, no coefficients of any predictors are shrunk to 0 (Figure 17 and Table 7). Thus, all 13 predictors are included in the model, which are age, chol, fbs1, sex, cp, trestbps, restecg, thalach, exang, oldpeak, slope, ca, and thal (Table 7).

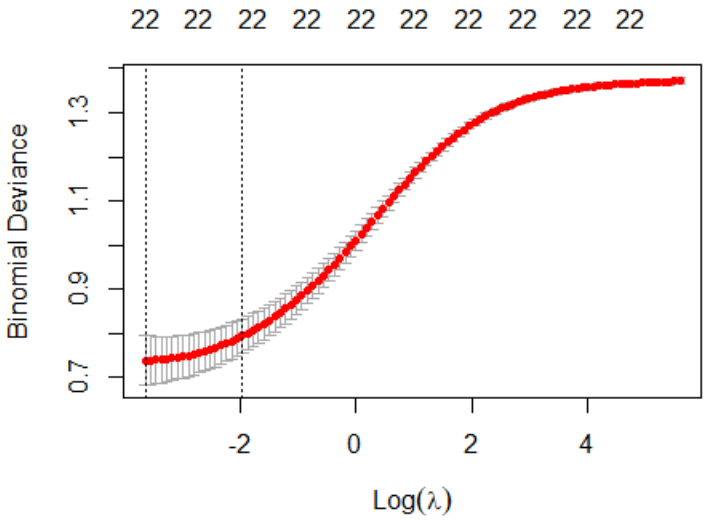


Figure 16. The deviance over different weighting parameter λ of Ridge.

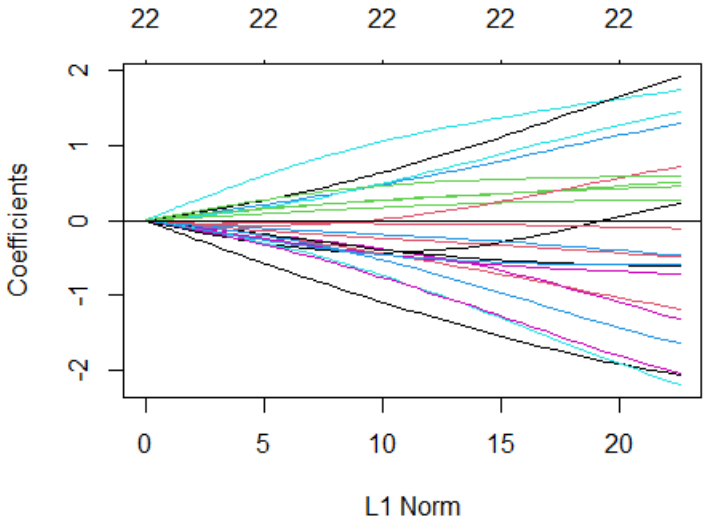


Figure 17. The shrinkage of coefficients for Ridge.

Table 7. The predictor variables selected by Ridge.

```
## (Intercept)  0.70016330
## age         -0.37813015
## sex1        -0.59760090
## cp1          0.32402819
## cp2          0.65184118
## cp3          0.71377466
## trestbps    -0.52461165
## chol        -0.47991164
## fbs1        -0.05000518
## restecg1     0.21249846
## restecg2    -0.23760952
## thalach      1.24615367
## exang1      -0.54262751
## oldpeak     -1.35980588
## slope1      -0.29011766
## slope2       0.31627667
## ca1         -0.77092762
## ca2         -1.05229251
## ca3         -1.05335435
## ca4          0.89947636
## thal1        0.13551235
## thal2        0.52149320
## thal3       -0.52930806
```

The Ridge model built on the train data with the selected predictors was then fitted to the test data. An AUC of 0.929 was achieved by ROC analysis (Figure 18), which is notably higher than that of the full, AIC, and BIC models, and is close to that of the LASSO model. By maximizing the Youden index, we obtained only one cutoff value (Table 8 and Figure 19). The corresponding pair of sensitivity and specificity at each cutoff is shown in Table 8.

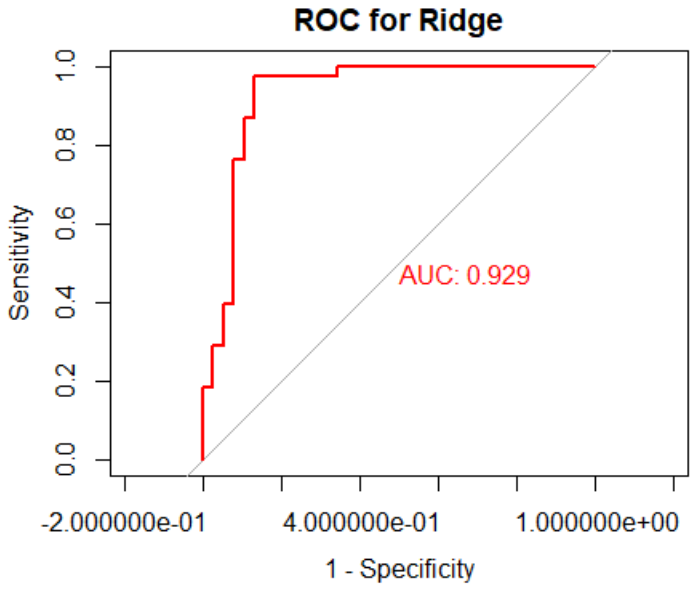


Figure 18. The ROC curve of Ridge logistic model on the test data.

Table 8. The classification metrics of the Ridge model using optimal Youden cutoffs.

```
##      Cutoffs Sensitivity Specificity    Youden
## 35 0.5038792   0.9736842   0.8684211 0.8421053
```

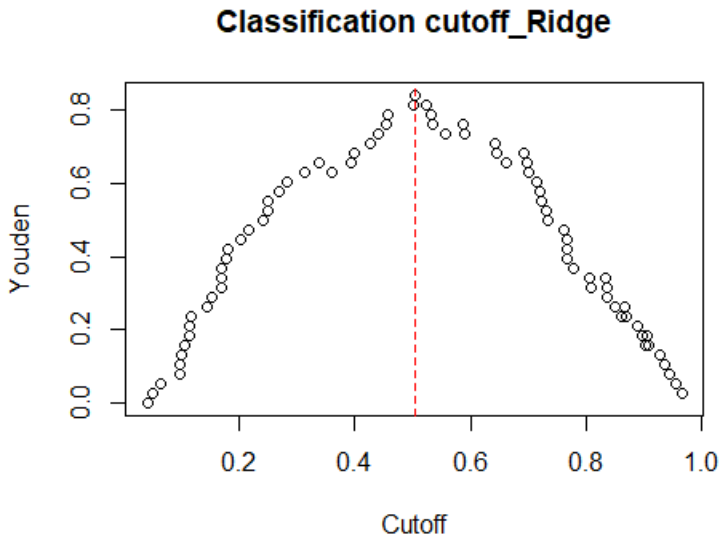


Figure 19. The Youden index at different cutoffs for the Ridge model on the test data.

2.2 kNN

An alternative to logistic regression is the kNN algorithm, which intends to predict based on the similar (close) records, i.e., k-nearest neighbors, on the train data. The tuning parameter to control the bias-variance tradeoff is the number of neighbors k, from which the most frequent response value is taken to make new predictions. A small k gives us a more complex and more flexible model, while a big k gives us a simpler and more restrictive model. Here, the leave-one-out-cross-validation (LOOCV) was adopted to achieve the primary goal of finding the optimal k.

We used the misclassification rate, i.e., 1 – accuracy, to assess the fit of our classification models. As shown in Figure 20, the lowest misclassification rate is estimated to be 0.01762115 when k equals to 5. The kNN model with the optimal k was then fitted to the test data. An AUC of 0.926 was achieved by ROC analysis (Figure 21), which is notably higher than that of the full, AIC, and BIC models, and is close to that of the LASSO and Ridge models.

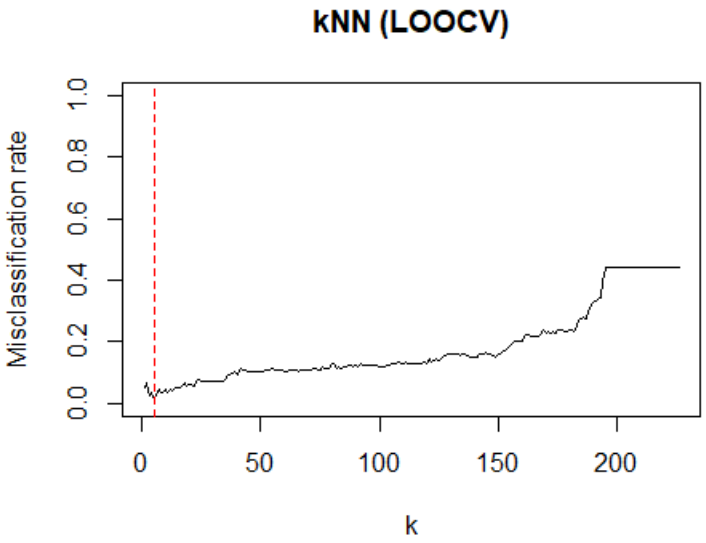


Figure 20. The misclassification rate at different k-nearest neighbors on the train data.

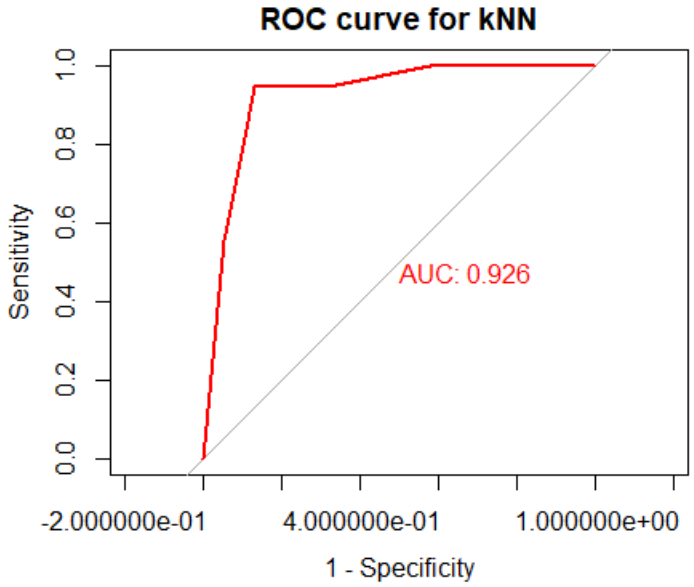


Figure 21. The ROC curve of kNN model on the test data.

2.3 Random Forest

We applied the Random Forest method on our classification task. Compared with the Bagging learning algorithm, the advantage of Random Forest is that it only allows to search a subset of features at each split, which will reduce the correlation among trees. The default number of features to be randomly selected is sqrt(p), which is 3 in our case. Therefore, we started with the default setting and evaluated its performance on test data.

With mtry = 3 and ntree = 500 (default), a RF model was fitted on the train set. By comparing the predicted probabilities of having heart disease and the true indicators, we generated an ROC curve (Figure 22) and the corresponding AUC = 0.887.

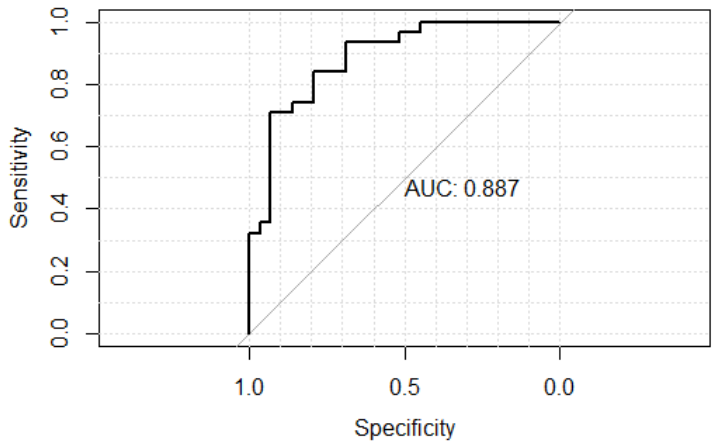


Figure 22.

An important parameter can be tuned in Random Forest is mtry. So we fixed ntree = 500, and let mtry vary from 1:10. We applied Repeated 5- folds Cross Validation and computed out-of-sample accuracies for various values of mtry. As the results shown in Figure 23, we concluded that mtry = 1 was preferred.

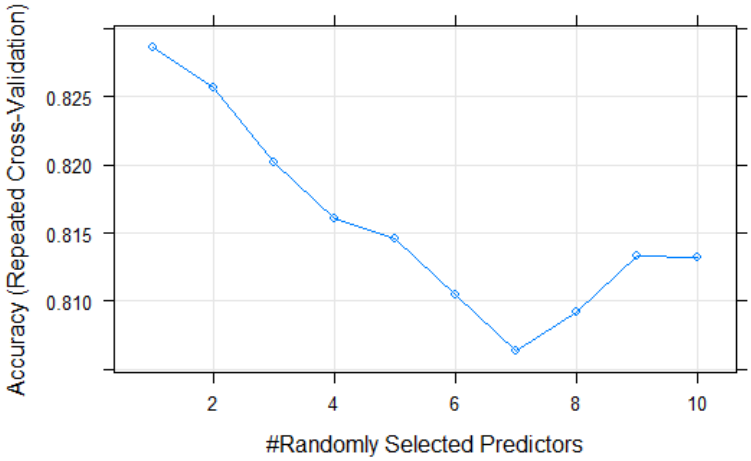


Figure 23.

For a more complete tuning process of Random Forest, we used grid search and cross validation to tune both ntree and mtry. Let ntree = 100, 500,1000 and mtry = 1:10, then we have 30 models in total. The highest CV accuracy was given by mtry = 1 and ntree = 1000 (See Appendix for full validation results), and we chose this model as our final decision.

Fit the final RF model on train data and obtain the AUC on test data. As shown in the ROC plot (Figure 24), our AUC has increased from 0.887 to 0.910.

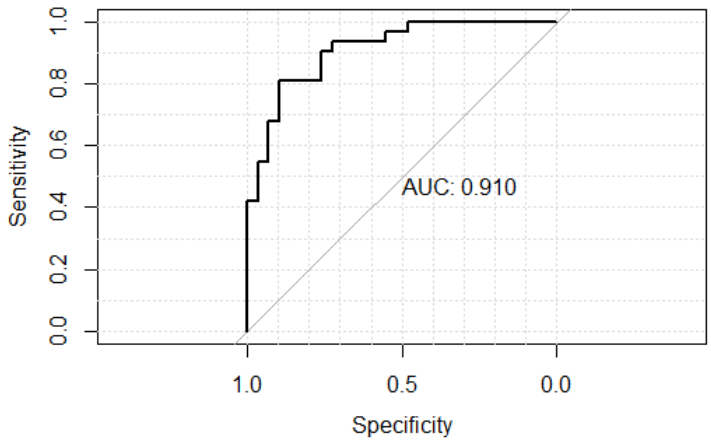


Figure 24.

Variable Importance Plot (Figure 25):

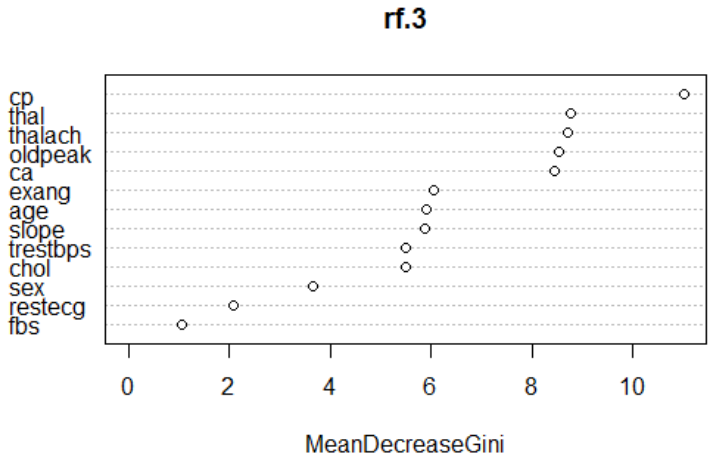


Figure 25.

2.4 Boosting

We performed Gradient Boosting using both gbm and xgboost. The parameters in gbm were carefully picked by grid search and validation, while xgboost was mainly used for comparison.

There are many parameters we can tune for gbm, but here I only picked three parameters that I cared most about. The tuning grid is shown in Table 9.

Table 9.

shrinkage	0.01	0.05	0.1
Interaction depth	1	2	3
ntree	500	1000	

Tuning method: Train - Validation

I used 75% of the train set to fit the model and used the remaining 25% data for validation. AUCs were computed for each candidate model based on validation data, and the one with highest AUC was selected as our final model.

Validation Results (see Appendix for full results):

The model with interaction.depth = 2, n.tree=500, shrinkage = 0.01 was selected, and its validation AUC was 0.9121.

Fit final model on the entire train set and make predictions on test data.

The ROC curve and AUC are shown in Figure 26:

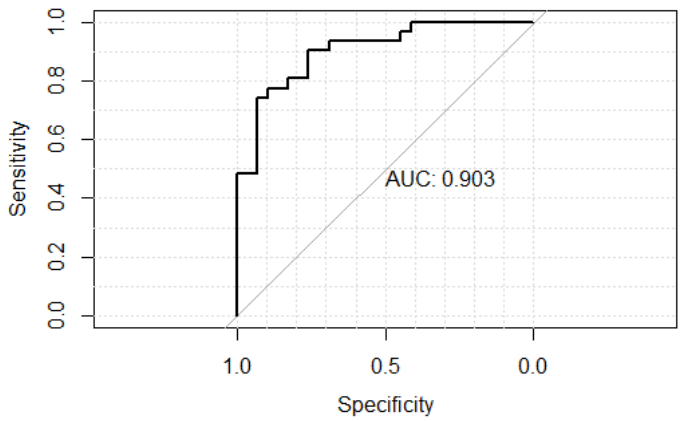


Figure 26.

This final AUC was slightly lower than that of Random Forest. We also reported the most important variables (Figure 27), and the top four were: cp, ca, thal, and oldpeak. This is consistent with the results from Random Forest.

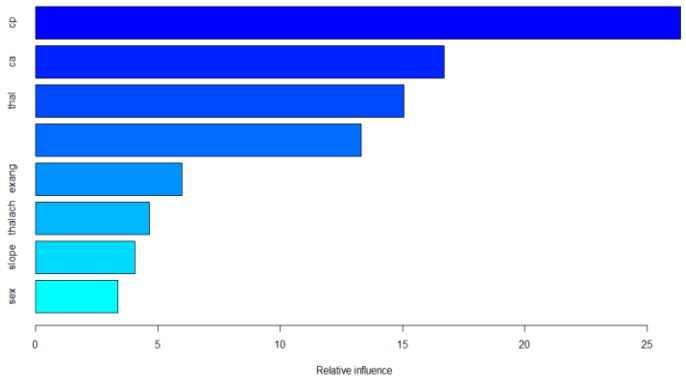


Figure 27.

For comparison, we ran boosting using xgboost and obtained AUC on test data which was 0.910. Therefore, boosting had the same performance with random forest.

The ROC Plot (Figure 28):

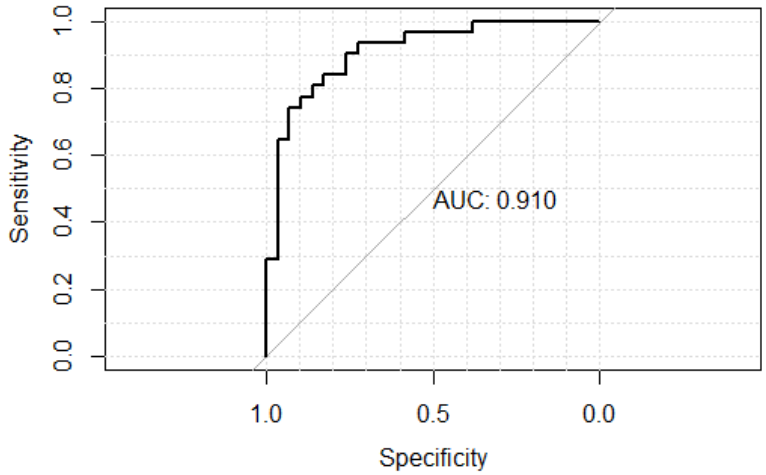


Figure 28.

2.5 Neural Networks

We constructed a neural network with two hidden layers and 16 units for each layer. Since this is a classification problem, a “sigmoid” activation was applied in the output layer. Moreover, we added a L2 penalty = 0.05 to neural nets. Fit the model on train data with epoch = 70 and predicted on test data. We observed that train loss and test loss were decreasing with epoch.

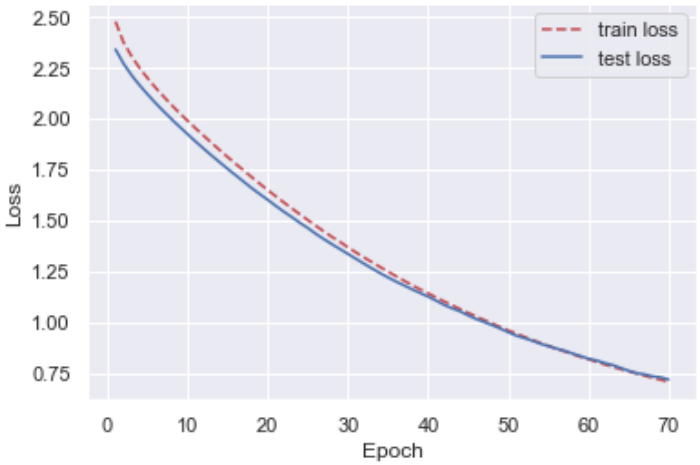


Figure 29.

To assess the performance of neural nets, we generated Lift Curve (Figure 30.) and ROC Curve (Figure 31.). The neural network model works well on test data with AUC attaining 0.935, which is the highest among all methods we have tried. As shown in the Lift Curve, about 50% of samples will get to 80% of positive cases.

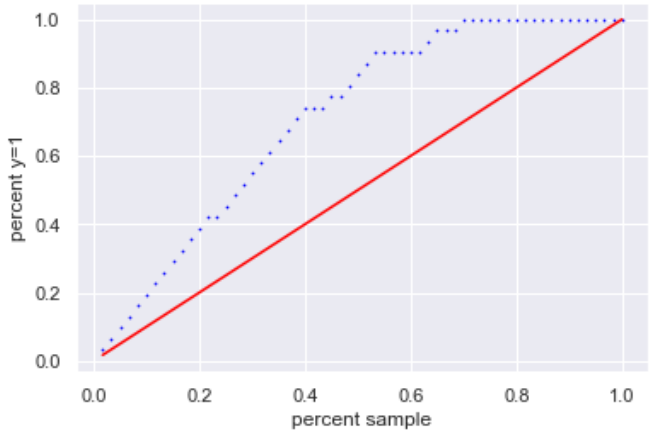


Figure 30.

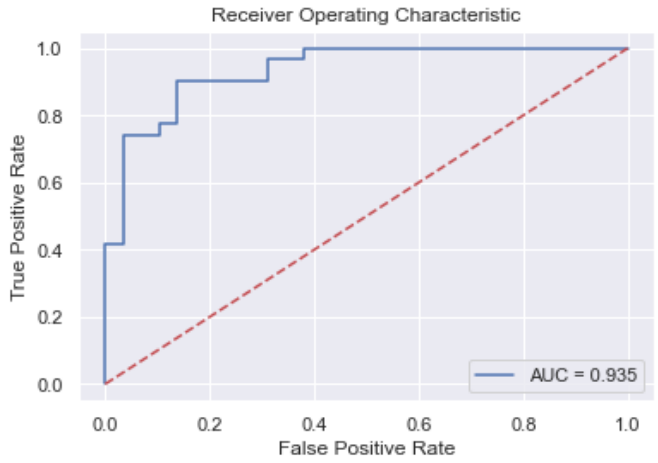


Figure 31.

3. Discussion and Conclusion

Table 10. The summary of methods and resulting parameters, AUC, and selected variables included in the models.

Method		Parameters	AUC on test data	Selected variables
Logistic regression	Full model	p = 13	0.870	All
	AIC	p = 8	0.868	sex cp trestbps exang oldpeak slope ca thal
	BIC	p = 6	0.862	sex cp trestbps exang slope ca
	LASSO	lambda min = 0.01233446 lambda 1se = 0.0284942	0.917	sex, cp, trestbps, restecg, thalach, exang, oldpeak, slope, ca, and thal
	Ridge	lambda min = 0.02657379 lambda 1se = 0.1418163	0.929	All
kNN		k = 5	0.926	
Random Forest, CV		Ntree = 1000 Mtry = 1	0.910	

Boosting, gbm	Interaction.depth = 2, ntree = 500, shrinkage = 0.01	0.903	
boosting , xgboost	max.depth = 2, eta = 0.01, nround = 500	0.910	
Neural networks	2 Layers ,16 units each layer, epoch = 70, L2 regularizer = 0.05	0.935	

The results table summarized the selected models for each method we have performed and their AUC on test data. The logistic regression was not competitive until ridge/lasso regularization was applied. The tree methods, Random Forest and Boosting are quite similar regarding the test AUC after tuning relevant parameters. The Neural Network shows the highest predictive ability, which is consistent with our expectation. However, it is worth noting that regularized logistic regression and kNN are also powerful classifiers for this heart disease data set.

Variable Selection Analysis

We have performed variable selection with logistic regression, and we also visualized variable influence while fitting tree models. Both random forest and boosting suggested that the crucial variables are cp, ca, thal, oldpeak and the least important variables are fbs, chol, age and restecg. However, when we performed lasso on logistic regression, the coefficients for age, fbs and chol were shrunk to zero. Therefore, these two methods give consistent results on variable selection that fbs, chol and age played less important roles in classification.

Pros and cons

1. Most of the machine learning algorithms require to normalize features before modeling, while this is not required by tree methods. Therefore, Random Forest and Boosting could be favored if we want to keep the data in original scales.
2. Boosting is more computationally efficient than Random Forest.
3. Logistic regression is more interpretable and simple compared with other learning algorithms. In our project, the regularized logistic regression is more powerful than boosting and similar with neural nets. In this circumstance, I may prefer logistic regression among all learning methods.

References

1. Murphy SL, Xu J, Kochanek KD, Arias E. Mortality in the United States, 2017. NCHS data brief, no 328. Hyattsville, MD: National Center for Health Statistics; 2018.
2. Symptoms, diagnosis and monitoring of arrhythmia. American Heart Association. https://www.heart.org/en/health-topics/arrhythmia/symptoms-diagnosis--monitoring-of-arrhythmia#.WQ0yvWnyt0w. Accessed Oct. 30, 2020.
3. Hungarian Institute of Cardiology. Budapest: Andras Janosi, M.D.
4. University Hospital, Zurich, Switzerland: William Steinbrunn, M.D.
5. University Hospital, Basel, Switzerland: Matthias Pfisterer, M.D.
6. V.A. Medical Center, Long Beach and Cleveland Clinic Foundation:Robert Detrano, M.D., Ph.D.

Appendix

```
rm(list=ls())
setwd("D:/ASU/Courses/STP 598_Machine Learning/Final project")
```

Load libraries and functions

```
library(rpart)
library(MASS)
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.0.5

library(randomForest)

## Warning: package 'randomForest' was built under R version 4.0.5

## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

library(gbm) # boosting

## Warning: package 'gbm' was built under R version 4.0.5

## Loaded gbm 2.1.8

library(caret)

## Warning: package 'caret' was built under R version 4.0.3

## Loading required package: lattice

## Warning: package 'lattice' was built under R version 4.0.3

## Loading required package: ggplot2

## Warning: package 'ggplot2' was built under R version 4.0.3

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:randomForest':
##
##     margin

library(kknn) # knn

## Warning: package 'kknn' was built under R version 4.0.3

##
## Attaching package: 'kknn'
```

```
## The following object is masked from 'package:caret':
##
##     contr.dummy
```

```
library(class) # knn.cv
```

```
## Warning: package 'class' was built under R version 4.0.5
```

```
library(pROC) # roc
```

```
## Warning: package 'pROC' was built under R version 4.0.3
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(OptimalCutpoints)
```

```
## Warning: package 'OptimalCutpoints' was built under R version 4.0.3
```

```
library(car)
```

```
## Loading required package: carData
```

```
library(mice)
```

```
## Warning: package 'mice' was built under R version 4.0.3
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
library(binomTools)
```

```
## Registered S3 method overwritten by 'binomTools':
##   method      from
##   profile.glm MASS
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.0.3
```

```
library(ggplot2)
library(reshape2) # melt

## Warning: package 'reshape2' was built under R version 4.0.5

library(leaps) # regsubsets

## Warning: package 'leaps' was built under R version 4.0.3

library(bestglm) # bestglm

## Warning: package 'bestglm' was built under R version 4.0.5

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.0.4

## Loading required package: Matrix

## Loaded glmnet 4.1-1

source("docv.R")
```

Read in data table and modify

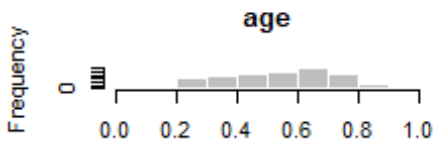
```
heart <- read.csv("heart.csv")

str(heart)

## 'data.frame':    303 obs. of  14 variables:
##  $ ï..age  : int  63 37 41 56 57 57 56 44 52 57 ...
##  $ sex     : int  1 1 0 1 0 1 0 1 1 1 ...
##  $ cp      : int  3 2 1 1 0 0 1 1 2 2 ...
##  $ trestbps: int  145 130 130 120 120 140 140 120 172 150 ...
##  $ chol    : int  233 250 204 236 354 192 294 263 199 168 ...
##  $ fbs     : int  1 0 0 0 0 0 0 0 1 0 ...
##  $ restecg : int  0 1 0 1 1 1 0 1 1 1 ...
##  $ thalach : int  150 187 172 178 163 148 153 173 162 174 ...
##  $ exang   : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ oldpeak : num  2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
##  $ slope   : int  0 0 2 2 2 1 1 2 2 2 ...
##  $ ca      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ thal    : int  1 2 2 2 2 1 2 3 3 2 ...
##  $ target  : int  1 1 1 1 1 1 1 1 1 1 ...

##### correct typo in original data table
colnames(heart)[1] <- c("age")
str(heart)

## 'data.frame':    303 obs. of  14 variables:
##  $ age     : int  63 37 41 56 57 57 56 44 52 57 ...
##  $ sex     : int  1 1 0 1 0 1 0 1 1 1 ...
##  $ cp      : int  3 2 1 1 0 0 1 1 2 2 ...
```

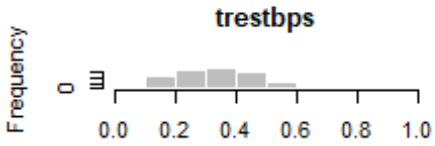
```
##  $ trestbps: int  145 130 130 120 120 140 140 120 172 150 ...
##  $ chol    : int  233 250 204 236 354 192 294 263 199 168 ...
##  $ fbs     : int  1 0 0 0 0 0 0 0 1 0 ...
##  $ restecg : int  0 1 0 1 1 1 0 1 1 1 ...
##  $ thalach : int  150 187 172 178 163 148 153 173 162 174 ...
##  $ exang   : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ oldpeak : num  2.3 3.5 1.4 0.8 0.6 0.4 1.3 0 0.5 1.6 ...
##  $ slope   : int  0 0 2 2 2 1 1 2 2 2 ...
##  $ ca      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ thal    : int  1 2 2 2 2 1 2 3 3 2 ...
##  $ target  : int  1 1 1 1 1 1 1 1 1 1 ...
```

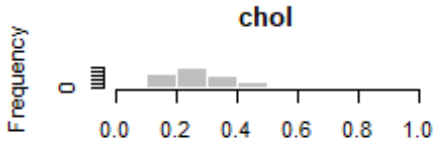
```
##### factorize categorical variables
heart$sex <- factor(heart$sex)
heart$cp <- factor(heart$cp)
heart$thal <- factor(heart$thal)
heart$fbs <- factor(heart$fbs)
heart$restecg <- factor(heart$restecg)
heart$exang <- factor(heart$exang)
heart$target <- factor(heart$target)
heart$slope <- factor(heart$slope)
heart$ca <- factor(heart$ca)


##### scale numerical variables
mmsc <- function(x) {return((x-min(x))/(max(x)-min(x)))} # scaling function

numv <- sapply(heart, is.numeric) # pick numerical variables
heart[numv] <- lapply(heart[numv], mmsc) # apply scaling function

attach(heart)
```

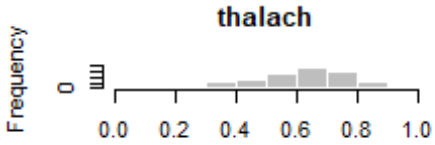
Check distribution of variables

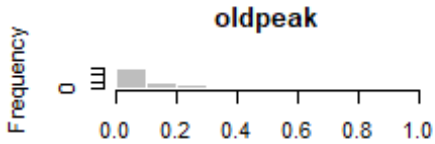
```
##### multiple histograms for numericals
par(mfrow = c(3, 2))
colname.numv <- dimnames(heart[numv])[[2]]
nnumv <- ncol(heart[numv])

for (i in 1:nnumv) {
  hist(heart[numv][,i], main = colname.numv[i], freq = TRUE,
       col = "gray", border = "white", xlab = "")
}


##### multiple barplots for categoricals
catv <- sapply(heart, is.factor) # pick categorical variables

par(mfrow = c(3, 3))
```



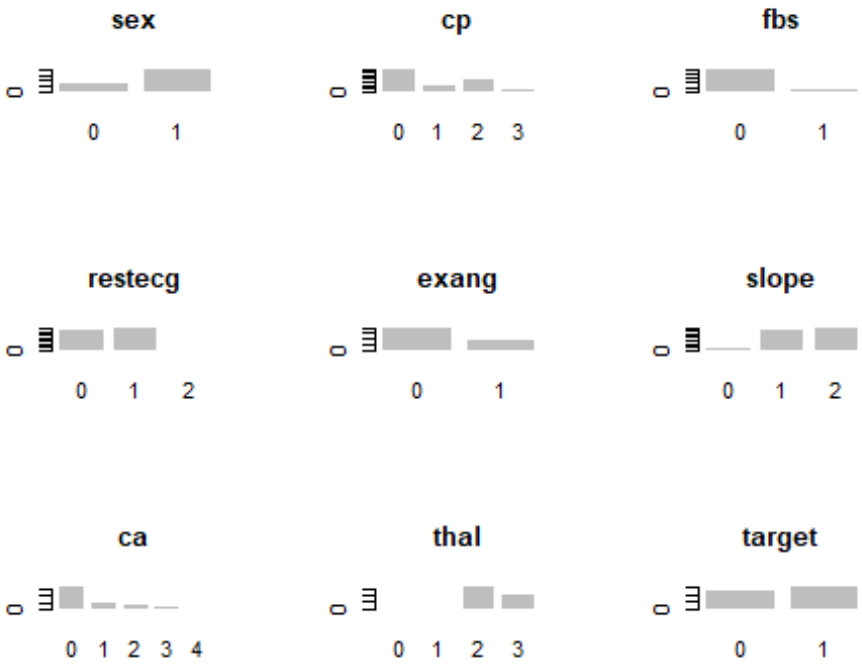




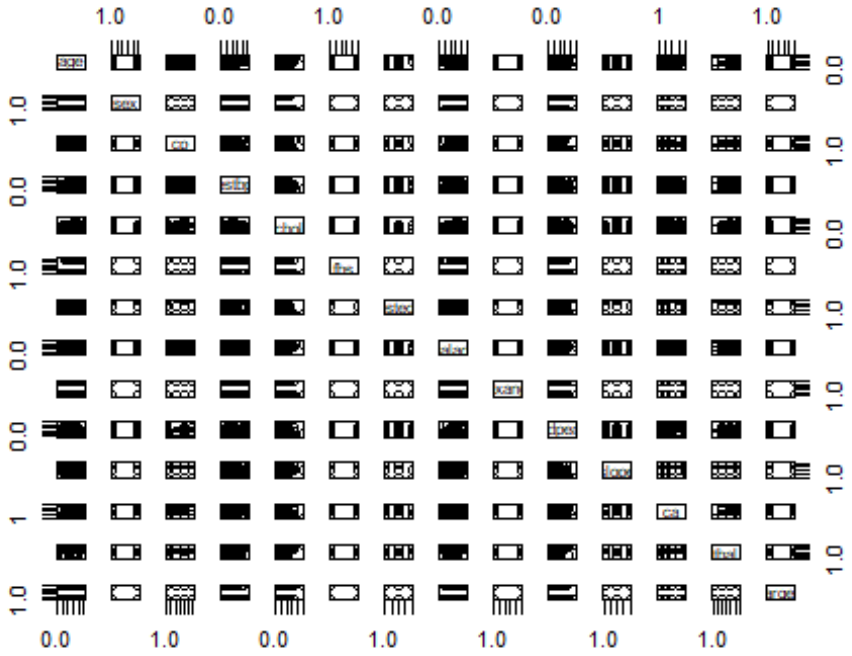





```
colname.catv <- dimnames(heart[catv])[[2]]
ncatv <- ncol(heart[catv])
for (i in 1:ncatv) {
  plot(heart[catv][,i], main = colname.catv[i], col = "gray",
       border = "white", xlab = "")
}
```




```
##### plot marginal relationship of all variables
pairs(heart)
```



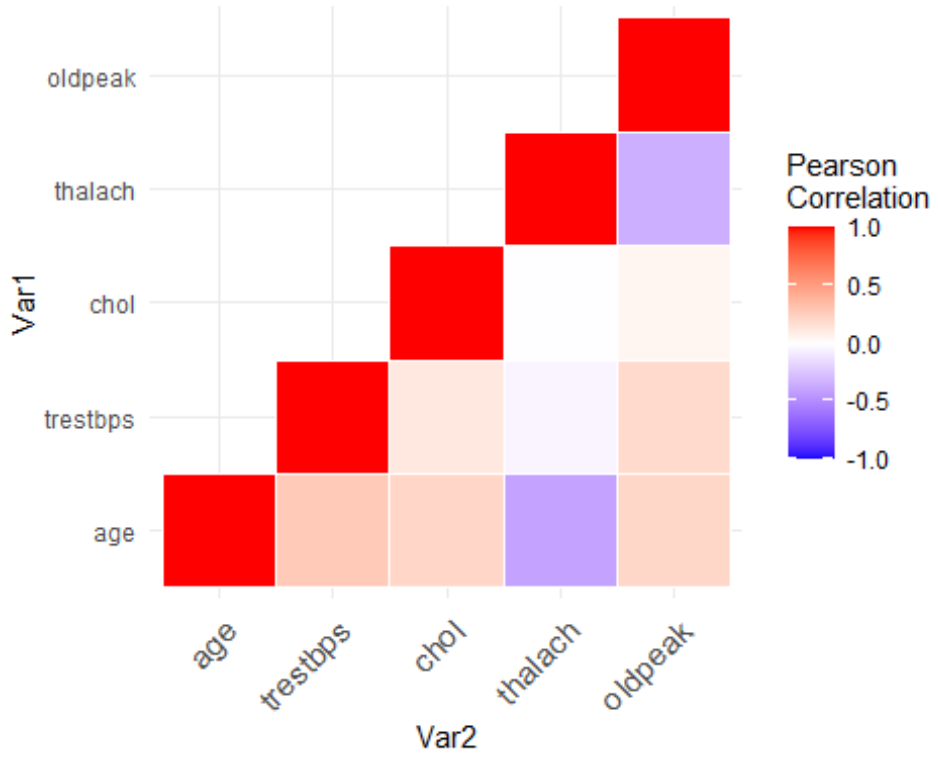
```r
##### check multicollinearity of numerical variables
cor(heart[sapply(heart, is.numeric)])
```

```
##                 age    trestbps         chol      thalach     oldpeak
## age       1.0000000  0.27935091  0.213677957 -0.398521938  0.21001257
## trestbps  0.2793509  1.00000000  0.123174207 -0.046697728  0.19321647
## chol      0.2136780  0.12317421  1.000000000 -0.009939839  0.05395192
## thalach  -0.3985219 -0.04669773 -0.009939839  1.000000000 -0.34418695
## oldpeak   0.2100126  0.19321647  0.053951920 -0.344186948  1.00000000
```

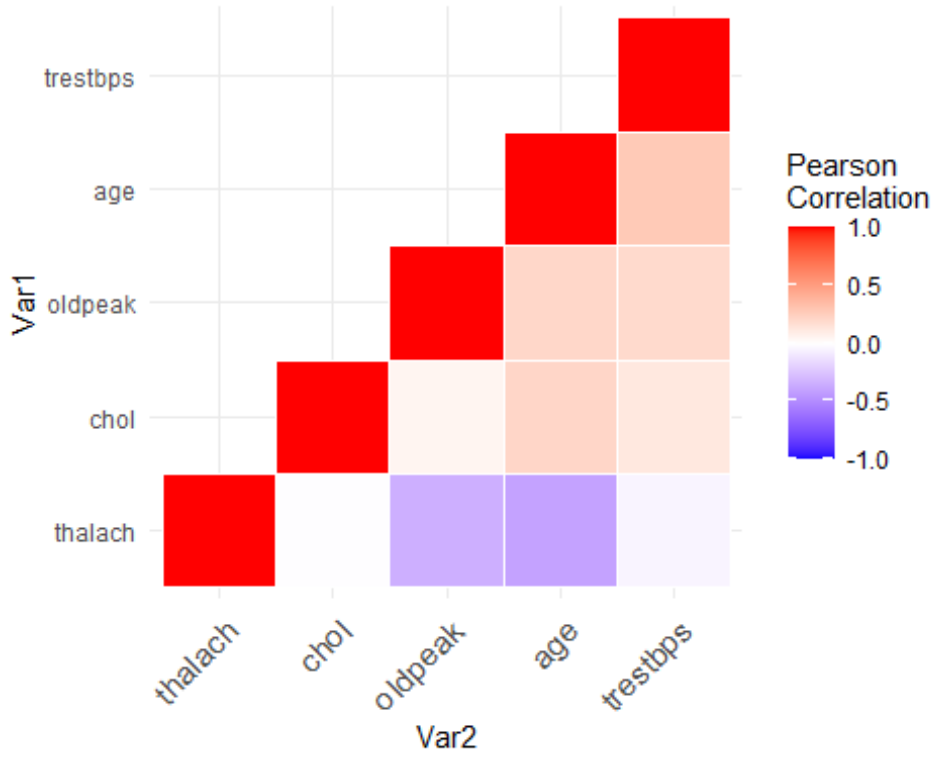
```r
# correlation matrix
cor.mat <- round(cor(heart[sapply(heart, is.numeric)]), 3)

up.tri <- function(cor.mat){
  cor.mat[lower.tri(cor.mat)]<- NA
  return(cor.mat)
}

upper.tri <- up.tri(cor.mat)
upper.tri
```

```
##          age trestbps  chol thalach oldpeak
## age        1    0.279 0.214  -0.399   0.210
## trestbps  NA    1.000 0.123  -0.047   0.193
## chol      NA       NA 1.000  -0.010   0.054
## thalach   NA       NA    NA   1.000  -0.344
## oldpeak   NA       NA    NA      NA   1.000
```

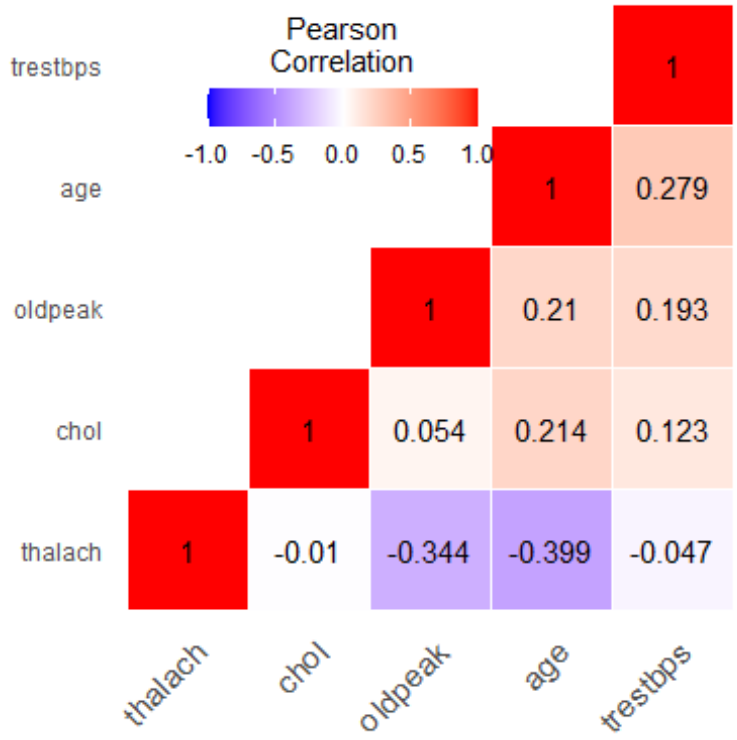
```r
melted.cor.mat <- melt(upper.tri, na.rm = TRUE)

ggplot(data = melted.cor.mat, aes(Var2, Var1, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "blue", high = "red",
                       mid = "white", midpoint = 0,
                       limit = c(-1, 1), space = "Lab",
                       name = "Pearson\nCorrelation") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                   size = 12, hjust = 1)) +
  coord_fixed()
```




```
reorder.cor.mat <- function(cor.mat){
  dd <- as.dist((1 - cor.mat) / 2)
  hc <- hclust(dd)
  cor.mat <- cor.mat[hc$order, hc$order]
}

cor.mat <- reorder.cor.mat(cor.mat)
upper.tri <- up.tri(cor.mat)

melted.cor.mat <- melt(upper.tri, na.rm = TRUE)

# plot heatmap of correlation matrix
ggheatmap <- ggplot(melted.cor.mat, aes(Var2, Var1,
                                        fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "blue", high = "red",
                       mid = "white", midpoint = 0,
                       limit = c(-1, 1), space = "Lab",
                       name = "Pearson\nCorrelation") +
  theme_minimal() + # minimal theme
  theme(axis.text.x = element_text(angle = 45, vjust = 1,
                                   size = 12, hjust = 1)) +
  coord_fixed()

print(ggheatmap)
```




```
# label correlation coefficients andlegend
ggheatmap +
  geom_text(aes(Var2, Var1, label = value), color = "black", size = 4) +
  theme(axis.title.x = element_blank(),
        axis.title.y = element_blank(),
        panel.grid.major = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank(),
        axis.ticks = element_blank(),
        legend.justification = c(1, 0),
        legend.position = c(0.6, 0.7),
        legend.direction = "horizontal") +
  guides(fill = guide_colorbar(barwidth = 7, barheight = 1,
                               title.position = "top",
                               title.hjust = 0.5))
```



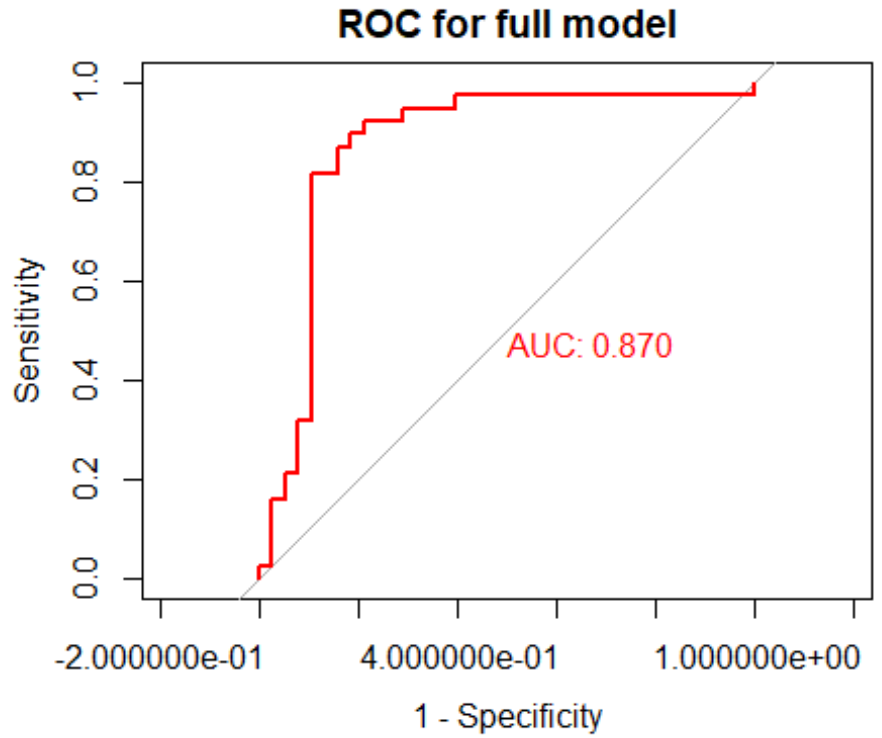
Split dataset into train and test

```
# set the seed to make partition reproducible
set.seed(101)

# 75% of data
smp.size <- floor(0.75 * nrow(heart))


##### select 75% of data as sample from total 'n' rows of the data
smp <- sample.int(n = nrow(heart), size = smp.size,
                  replace = FALSE)
train <- heart[smp, ]
test <- heart[-smp, ]
```

Method 1: logistic regression 1.1. Full model

```
##### fit the full model on train set
set.seed(123)
full.fit <- glm(target ~ ., data = train, family = binomial)


##### predict on test set
pi.full <- predict(full.fit, newdata = test, type = "response")

# plot ROC curve and AUC
roc.full <- roc(target ~ pi.full, data = test, plot = TRUE,
```

```
                legacy.axes = TRUE, col = "red", lwd = 2,
                main = "ROC for full model", print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
# find the optimal cutoff for classification
roc.full.y <- data.frame(pi.full, observed.class = test$target)
full.opt <- optimal.cutpoints(X = "pi.full",
                              status = "observed.class",
                              tag.healthy = 0,
                              methods = "Youden",
                              data = roc.full.y)
# other methods include "MaxEfficiency" for max accuracy, similar to min erro
r rate
full.opt # # 2 optimal cutoffs give same Youden value

##
## Call:
## optimal.cutpoints.default(X = "pi.full", status = "observed.class",
##     tag.healthy = 0, methods = "Youden", data = roc.full.y)
##
## Optimal cutoffs:
##   Youden
## 1 0.4028
## 2 0.4239
```

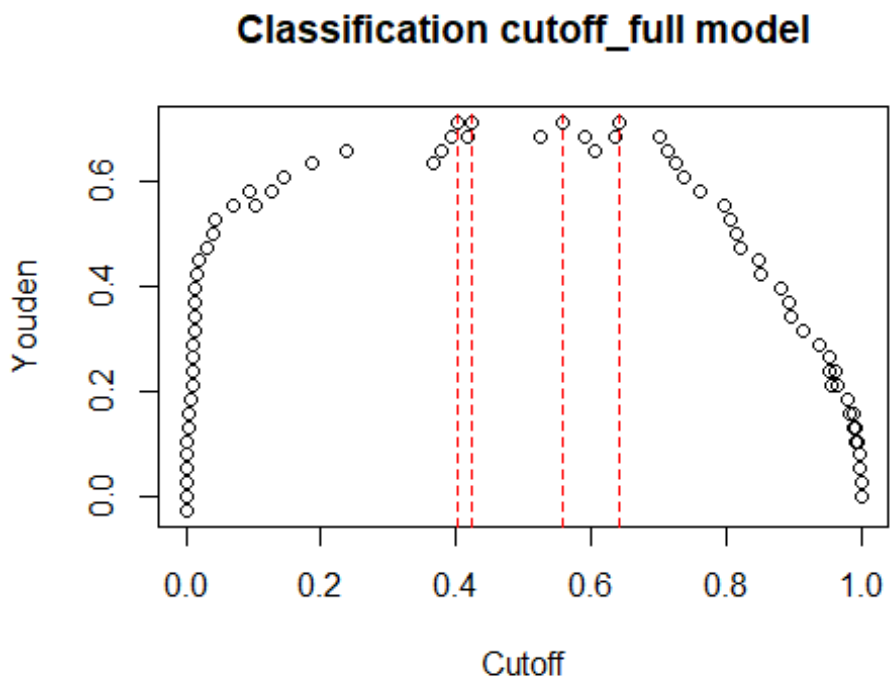
```
## 3 0.5585
## 4 0.6406
##
## Area under the ROC curve (AUC):  0.87 (0.776, 0.964)
```

```
# find sensitivity and specificity under optimal cutoff
full.tmp <- full.opt$Youden$Global$measures.acc
full.youden <- data.frame(full.tmp$cutoffs, full.tmp$Se,
                          full.tmp$Sp,
                          full.tmp$Se + full.tmp$Sp - 1)
colnames(full.youden) <- c("Cutoffs", "Sensitivity",
                           "Specificity", "Youden")
full.youden # all possible cutoffs
```

```
##         Cutoffs Sensitivity Specificity      Youden
## 1  4.276553e-05  1.00000000  0.00000000  0.00000000
## 2  3.462294e-04  0.97368421  0.00000000 -0.02631579
## 3  5.020297e-04  0.97368421  0.02631579  0.00000000
## 4  1.368491e-03  0.97368421  0.05263158  0.02631579
## 5  1.503701e-03  0.97368421  0.07894737  0.05263158
## 6  2.051799e-03  0.97368421  0.10526316  0.07894737
## 7  2.166695e-03  0.97368421  0.13157895  0.10526316
## 8  4.089534e-03  0.97368421  0.15789474  0.13157895
## 9  4.259918e-03  0.97368421  0.18421053  0.15789474
## 10 8.005947e-03  0.97368421  0.21052632  0.18421053
## 11 8.968946e-03  0.97368421  0.23684211  0.21052632
## 12 9.039114e-03  0.97368421  0.26315789  0.23684211
## 13 9.310002e-03  0.97368421  0.28947368  0.26315789
## 14 1.004308e-02  0.97368421  0.31578947  0.28947368
## 15 1.177710e-02  0.97368421  0.34210526  0.31578947
## 16 1.334926e-02  0.97368421  0.36842105  0.34210526
## 17 1.368339e-02  0.97368421  0.39473684  0.36842105
## 18 1.468160e-02  0.97368421  0.42105263  0.39473684
## 19 1.609571e-02  0.97368421  0.44736842  0.42105263
## 20 2.053855e-02  0.97368421  0.47368421  0.44736842
## 21 3.184519e-02  0.97368421  0.50000000  0.47368421
## 22 4.163537e-02  0.97368421  0.52631579  0.50000000
## 23 4.256498e-02  0.97368421  0.55263158  0.52631579
## 24 6.880929e-02  0.97368421  0.57894737  0.55263158
## 25 9.425931e-02  0.97368421  0.60526316  0.57894737
## 26 1.029170e-01  0.94736842  0.60526316  0.55263158
## 27 1.268040e-01  0.94736842  0.63157895  0.57894737
## 28 1.463654e-01  0.94736842  0.65789474  0.60526316
## 29 1.859702e-01  0.94736842  0.68421053  0.63157895
## 30 2.369208e-01  0.94736842  0.71052632  0.65789474
## 31 3.675306e-01  0.92105263  0.71052632  0.63157895
## 32 3.784055e-01  0.92105263  0.73684211  0.65789474
## 33 3.944360e-01  0.92105263  0.76315789  0.68421053
## 34 4.028204e-01  0.92105263  0.78947368  0.71052632
## 35 4.175277e-01  0.89473684  0.78947368  0.68421053
```

```
## 36 4.239047e-01  0.89473684  0.81578947  0.71052632
## 37 5.252152e-01  0.86842105  0.81578947  0.68421053
## 38 5.584640e-01  0.86842105  0.84210526  0.71052632
## 39 5.901100e-01  0.84210526  0.84210526  0.68421053
## 40 6.046116e-01  0.81578947  0.84210526  0.65789474
## 41 6.358447e-01  0.81578947  0.86842105  0.68421053
## 42 6.406056e-01  0.81578947  0.89473684  0.71052632
## 43 7.013028e-01  0.78947368  0.89473684  0.68421053
## 44 7.137199e-01  0.76315789  0.89473684  0.65789474
## 45 7.263111e-01  0.73684211  0.89473684  0.63157895
## 46 7.361623e-01  0.71052632  0.89473684  0.60526316
## 47 7.602274e-01  0.68421053  0.89473684  0.57894737
## 48 7.975757e-01  0.65789474  0.89473684  0.55263158
## 49 8.061737e-01  0.63157895  0.89473684  0.52631579
## 50 8.143383e-01  0.60526316  0.89473684  0.50000000
## 51 8.216274e-01  0.57894737  0.89473684  0.47368421
## 52 8.469721e-01  0.55263158  0.89473684  0.44736842
## 53 8.507440e-01  0.52631579  0.89473684  0.42105263
## 54 8.823571e-01  0.50000000  0.89473684  0.39473684
## 55 8.930113e-01  0.47368421  0.89473684  0.36842105
## 56 8.958680e-01  0.44736842  0.89473684  0.34210526
## 57 9.144149e-01  0.42105263  0.89473684  0.31578947
## 58 9.377692e-01  0.39473684  0.89473684  0.28947368
## 59 9.516958e-01  0.36842105  0.89473684  0.26315789
## 60 9.518274e-01  0.34210526  0.89473684  0.23684211
## 61 9.566698e-01  0.31578947  0.89473684  0.21052632
## 62 9.616868e-01  0.31578947  0.92105263  0.23684211
## 63 9.658604e-01  0.28947368  0.92105263  0.21052632
## 64 9.806503e-01  0.26315789  0.92105263  0.18421053
## 65 9.828281e-01  0.23684211  0.92105263  0.15789474
## 66 9.880732e-01  0.21052632  0.92105263  0.13157895
## 67 9.895970e-01  0.21052632  0.94736842  0.15789474
## 68 9.900198e-01  0.18421053  0.94736842  0.13157895
## 69 9.912526e-01  0.15789474  0.94736842  0.10526316
## 70 9.916209e-01  0.15789474  0.97368421  0.13157895
## 71 9.934000e-01  0.13157895  0.97368421  0.10526316
## 72 9.986804e-01  0.10526316  0.97368421  0.07894737
## 73 9.988506e-01  0.07894737  0.97368421  0.05263158
## 74 9.991568e-01  0.05263158  0.97368421  0.02631579
## 75 9.999968e-01  0.02631579  0.97368421  0.00000000
## 76 1.000000e+00  0.02631579  1.00000000  0.02631579
```

```
# plot cutoffs vs Youden
plot(full.youden$Cutoffs, full.youden$Youden,
     xlab = "Cutoff", ylab = "Youden",
     main = "Classification cutoff_full model")
abline(v = full.opt[[1]]$Global$optimal.cutoff$cutoff,
       col = "red", lty = 2) # cutoffs for max Youden
```




```
full.youden.opt <- full.youden[which(full.youden$Youden
                                     == max(full.youden$Youden)), ]
full.youden.opt # sens and spec under optimal cutoff
```

```
##      Cutoffs Sensitivity Specificity    Youden
## 34 0.4028204   0.9210526   0.7894737 0.7105263
## 36 0.4239047   0.8947368   0.8157895 0.7105263
## 38 0.5584640   0.8684211   0.8421053 0.7105263
## 42 0.6406056   0.8157895   0.8947368 0.7105263
```

1.2. AIC

```
# the outcome variable must be named y, no extraneous variables should be pre
sent in the dataset.
train2 <- within(train, {
    y    <- target         # rename target as y
    target  <- NULL        # delete target
})

test2 <- within(test, {
    y    <- target         # rename target as y
    target  <- NULL        # delete target
})


##### select subset with AIC
set.seed(123)
```

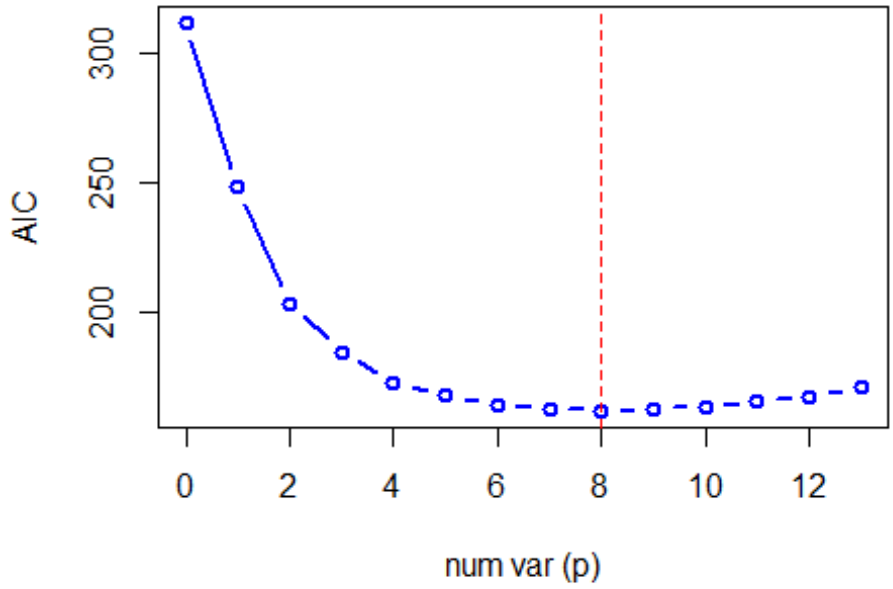
```r
p <- ncol(train) - 1 # number of variables to try
aic.bestglm <- bestglm(Xy = train2, family = binomial,
                       IC = "AIC", method = "exhaustive",
                       nvmax = p) # nvmax = number of variables allowed to tr
y

## Morgan-Tatar search since family is non-gaussian.

## Note: factors present with more than 2 levels.

summary.bestglm(aic.bestglm)

## Fitting algorithm:  AIC-glm
## Best Model:
##             df deviance
## Null Model 210 129.5880
## Full Model 226 311.4698
##
##  likelihood-ratio test - GLM
##
## data:  H0: Null Model vs. H1: Best Fit AIC-glm
## X = 181.88, df = 16, p-value < 2.2e-16

# check selected variables in top 5 models
aic.bestglm$BestModels # see which variables are in the selected subset of ea
ch model

##     age  sex   cp trestbps  chol   fbs restecg thalach exang oldpeak slope
   ca
## 1 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE   FALSE  TRUE    TRUE  TRUE
 TRUE
## 2 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE    TRUE  TRUE    TRUE  TRUE
 TRUE
## 3 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE   FALSE FALSE    TRUE  TRUE
 TRUE
## 4 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE    TRUE  TRUE   FALSE  TRUE
 TRUE
## 5 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE   FALSE  TRUE   FALSE  TRUE
 TRUE
##   thal Criterion
## 1 TRUE  161.5880
## 2 TRUE  162.3596
## 3 TRUE  162.5996
## 4 TRUE  162.6263
## 5 TRUE  162.6527

##### determine the best p for AIC
aic.reg <- as.numeric(unlist(aic.bestglm$Subsets["AIC"]))
aic.reg # AIC of subset models
```

```
##  [1] 311.4698 248.0411 203.0682 184.5253 172.5462 167.6511 164.0427 162.59
96
##  [9] 161.5880 162.3596 163.3187 165.1403 167.0976 170.6246
```

```
p.aicbest <- which.min(aic.reg) - 1 # p for smallest AIC
cat("The best p for subset with smallest AIC is", p.aicbest, "\n")
```

```
## The best p for subset with smallest AIC is 8
```

```
# plot AIC vs p for each model
plot(x = c(0:13), y = aic.reg, xlab = "num var (p)", ylab = "AIC",
     type = 'b', col = "blue", lwd = 2, cex.lab = 1)
abline(v = p.aicbest, col = "red", lty = 2)
```

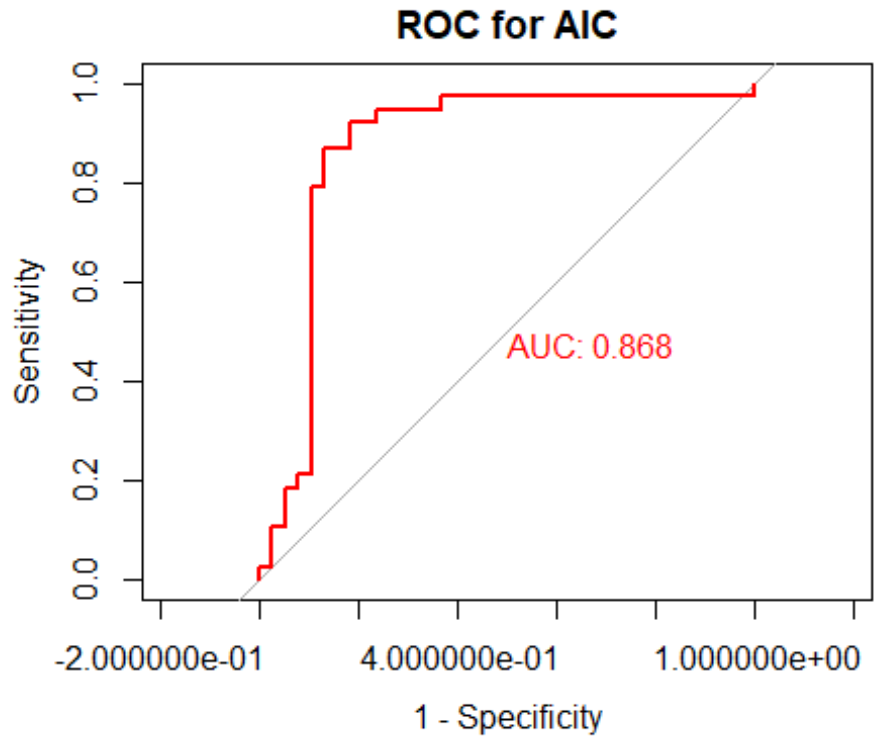



```
##### get the selected variables in best model
aic.best.v <- aic.bestglm$BestModels[which.min(aic.bestglm$BestModels$Criteri
on),]
aic.best.v # the best model
```

```
##     age  sex   cp trestbps  chol   fbs restecg thalach exang oldpeak slope
   ca
## 1 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE   FALSE  TRUE    TRUE  TRUE
 TRUE
##   thal Criterion
## 1 TRUE   161.588
```

```r
aic.sv <- unique(colnames(aic.best.v)[which(aic.best.v == "TRUE", arr.ind = T
RUE)[,2]])
cat("The selected variables for best model with AIC are:", aic.sv, "\n")

## The selected variables for best model with AIC are: sex cp trestbps exang
oldpeak slope ca thal

##### fit the selected variables on train set
aic.fit <- glm(y ~ sex + cp + trestbps + exang + oldpeak + slope
               + ca + thal, data = train2, family = binomial)
summary(aic.fit)

##
## Call:
## glm(formula = y ~ sex + cp + trestbps + exang + oldpeak + slope +
##     ca + thal, family = binomial, data = train2)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.88473  -0.24765   0.09156   0.40521   3.12931
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -11.4056  2399.5452  -0.005  0.99621
## sex1          -2.0685     0.6755  -3.062  0.00220 **
## cp1            0.6298     0.6597   0.955  0.33975
## cp2            2.3005     0.6316   3.643  0.00027 ***
## cp3            2.6138     0.9323   2.804  0.00505 **
## trestbps      -2.5507     1.4255  -1.789  0.07356 .
## exang1        -0.9232     0.5302  -1.741  0.08167 .
## oldpeak       -2.9452     1.7304  -1.702  0.08875 .
## slope1        -0.8756     0.8953  -0.978  0.32807
## slope2         1.0319     0.9954   1.037  0.29990
## ca1           -2.8207     0.6234  -4.525 6.05e-06 ***
## ca2           -3.5879     0.9147  -3.923 8.76e-05 ***
## ca3           -3.6605     1.4085  -2.599  0.00936 **
## ca4           15.5949  1118.5138   0.014  0.98888
## thal1         16.2468  2399.5449   0.007  0.99460
## thal2         15.3937  2399.5448   0.006  0.99488
## thal3         14.1179  2399.5448   0.006  0.99531
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 311.47  on 226  degrees of freedom
## Residual deviance: 129.59  on 210  degrees of freedom
## AIC: 163.59
##
## Number of Fisher Scoring iterations: 15
```

```
##### predict on test set
pi.aic <- predict(aic.fit, newdata = test2, type = "response")

# plot ROC curve and AUC
roc.aic <- roc(y ~ pi.aic, data = test2, plot = TRUE,
               legacy.axes = TRUE, col = "red", lwd = 2,
               main = "ROC for AIC",
               print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
# find the optimal cutoff for classification
roc.aic.y <- data.frame(pi.aic, observed.class = test2$y)
aic.opt <- optimal.cutpoints(X = "pi.aic",
                             status = "observed.class",
                             tag.healthy = 0,
                             methods = "Youden",
                             data = roc.aic.y)
# other methods include "MaxEfficiency" for max accuracy, similar to min erro
r rate
aic.opt # # 2 optimal cutoffs give same Youden value

##
## Call:
## optimal.cutpoints.default(X = "pi.aic", status = "observed.class",
```

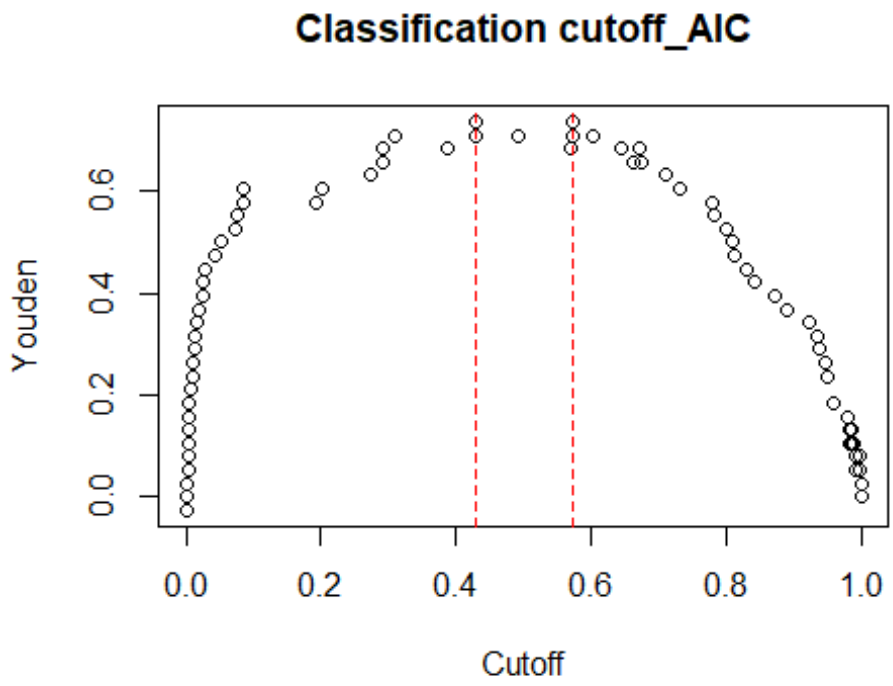
```
##     tag.healthy = 0, methods = "Youden", data = roc.aic.y)
##
## Optimal cutoffs:
##   Youden
## 1 0.4287
## 2 0.5739
##
## Area under the ROC curve (AUC):  0.868 (0.772, 0.965)
```

```
# find sensitivity and specificity under optimal cutoff
aic.tmp <- aic.opt$Youden$Global$measures.acc
aic.youden <- data.frame(aic.tmp$cutoffs, aic.tmp$Se,
                         aic.tmp$Sp,
                         aic.tmp$Se + aic.tmp$Sp - 1)
colnames(aic.youden) <- c("Cutoffs", "Sensitivity",
                          "Specificity", "Youden")
aic.youden # all possible cutoffs
```

```
##         Cutoffs Sensitivity Specificity      Youden
## 1  0.0001375479  1.00000000  0.00000000  0.00000000
## 2  0.0003641870  0.97368421  0.00000000 -0.02631579
## 3  0.0006749615  0.97368421  0.02631579  0.00000000
## 4  0.0026151759  0.97368421  0.05263158  0.02631579
## 5  0.0028993690  0.97368421  0.07894737  0.05263158
## 6  0.0031962199  0.97368421  0.10526316  0.07894737
## 7  0.0034671382  0.97368421  0.13157895  0.10526316
## 8  0.0041989348  0.97368421  0.15789474  0.13157895
## 9  0.0046367724  0.97368421  0.18421053  0.15789474
## 10 0.0056483298  0.97368421  0.21052632  0.18421053
## 11 0.0062343495  0.97368421  0.23684211  0.21052632
## 12 0.0093718076  0.97368421  0.26315789  0.23684211
## 13 0.0095570003  0.97368421  0.28947368  0.26315789
## 14 0.0128294225  0.97368421  0.31578947  0.28947368
## 15 0.0129475875  0.97368421  0.34210526  0.31578947
## 16 0.0177088469  0.97368421  0.36842105  0.34210526
## 17 0.0204699783  0.97368421  0.39473684  0.36842105
## 18 0.0247139537  0.97368421  0.42105263  0.39473684
## 19 0.0253645787  0.97368421  0.44736842  0.42105263
## 20 0.0276558194  0.97368421  0.47368421  0.44736842
## 21 0.0437368484  0.97368421  0.50000000  0.47368421
## 22 0.0534867653  0.97368421  0.52631579  0.50000000
## 23 0.0730576960  0.97368421  0.55263158  0.52631579
## 24 0.0765603968  0.97368421  0.57894737  0.55263158
## 25 0.0852829581  0.97368421  0.60526316  0.57894737
## 26 0.0855547137  0.97368421  0.63157895  0.60526316
## 27 0.1937269009  0.94736842  0.63157895  0.57894737
## 28 0.2011219609  0.94736842  0.65789474  0.60526316
## 29 0.2745036019  0.94736842  0.68421053  0.63157895
## 30 0.2907343580  0.94736842  0.71052632  0.65789474
## 31 0.2918075587  0.94736842  0.73684211  0.68421053
```

```
## 32 0.3103541545  0.94736842  0.76315789  0.71052632
## 33 0.3885528541  0.92105263  0.76315789  0.68421053
## 34 0.4283985356  0.92105263  0.78947368  0.71052632
## 35 0.4287411678  0.92105263  0.81578947  0.73684211
## 36 0.4921718894  0.89473684  0.81578947  0.71052632
## 37 0.5695636980  0.86842105  0.81578947  0.68421053
## 38 0.5727476335  0.86842105  0.84210526  0.71052632
## 39 0.5739492405  0.86842105  0.86842105  0.73684211
## 40 0.6011978010  0.84210526  0.86842105  0.71052632
## 41 0.6453762880  0.81578947  0.86842105  0.68421053
## 42 0.6615163202  0.78947368  0.86842105  0.65789474
## 43 0.6709620997  0.78947368  0.89473684  0.68421053
## 44 0.6737607362  0.76315789  0.89473684  0.65789474
## 45 0.7091878561  0.73684211  0.89473684  0.63157895
## 46 0.7312134974  0.71052632  0.89473684  0.60526316
## 47 0.7778352668  0.68421053  0.89473684  0.57894737
## 48 0.7815259920  0.65789474  0.89473684  0.55263158
## 49 0.7991786347  0.63157895  0.89473684  0.52631579
## 50 0.8083340941  0.60526316  0.89473684  0.50000000
## 51 0.8113048710  0.57894737  0.89473684  0.47368421
## 52 0.8312335063  0.55263158  0.89473684  0.44736842
## 53 0.8428847523  0.52631579  0.89473684  0.42105263
## 54 0.8725590271  0.50000000  0.89473684  0.39473684
## 55 0.8898666258  0.47368421  0.89473684  0.36842105
## 56 0.9236568520  0.44736842  0.89473684  0.34210526
## 57 0.9358577213  0.42105263  0.89473684  0.31578947
## 58 0.9379068395  0.39473684  0.89473684  0.28947368
## 59 0.9464366978  0.36842105  0.89473684  0.26315789
## 60 0.9505295508  0.34210526  0.89473684  0.23684211
## 61 0.9586887978  0.28947368  0.89473684  0.18421053
## 62 0.9807988029  0.26315789  0.89473684  0.15789474
## 63 0.9813157351  0.23684211  0.89473684  0.13157895
## 64 0.9816062729  0.21052632  0.89473684  0.10526316
## 65 0.9844053855  0.21052632  0.92105263  0.13157895
## 66 0.9845369133  0.18421053  0.92105263  0.10526316
## 67 0.9861830720  0.18421053  0.94736842  0.13157895
## 68 0.9879884702  0.15789474  0.94736842  0.10526316
## 69 0.9913346092  0.13157895  0.94736842  0.07894737
## 70 0.9927186055  0.10526316  0.94736842  0.05263158
## 71 0.9982280378  0.10526316  0.97368421  0.07894737
## 72 0.9987642463  0.07894737  0.97368421  0.05263158
## 73 0.9990738913  0.05263158  0.97368421  0.02631579
## 74 0.9999985987  0.02631579  0.97368421  0.00000000
## 75 0.9999999975  0.02631579  1.00000000  0.02631579

# plot cutoffs vs Youden
plot(aic.youden$Cutoffs, aic.youden$Youden,
     xlab = "Cutoff", ylab = "Youden",
     main = "Classification cutoff_AIC")
```

```
abline(v = aic.opt[[1]]$Global$optimal.cutoff$cutoff,
       col = "red", lty = 2) # cutoffs for max Youden
```




```
aic.youden.opt <- aic.youden[which(aic.youden$Youden
                                   == max(aic.youden$Youden)), ]
aic.youden.opt # sens and spec under optimal cutoff
```

```
##      Cutoffs Sensitivity Specificity    Youden
## 35 0.4287412   0.9210526   0.8157895 0.7368421
## 39 0.5739492   0.8684211   0.8684211 0.7368421
```

1.3. BIC

```
##### select subset with BIC
set.seed(123)
bic.bestglm <- bestglm(Xy = train2, family = binomial,
                       IC = "BIC", method = "exhaustive",
                       nvmax = p) # nvmax = number of variables allowed to tr
y

## Morgan-Tatar search since family is non-gaussian.

## Note: factors present with more than 2 levels.

summary.bestglm(bic.bestglm)

## Fitting algorithm:  BIC-glm
## Best Model:
```

```
##             df deviance
## Null Model 214 141.6571
## Full Model 226 311.4698
##
##  likelihood-ratio test - GLM
##
## data:  H0: Null Model vs. H1: Best Fit BIC-glm
## X = 169.81, df = 12, p-value < 2.2e-16
```

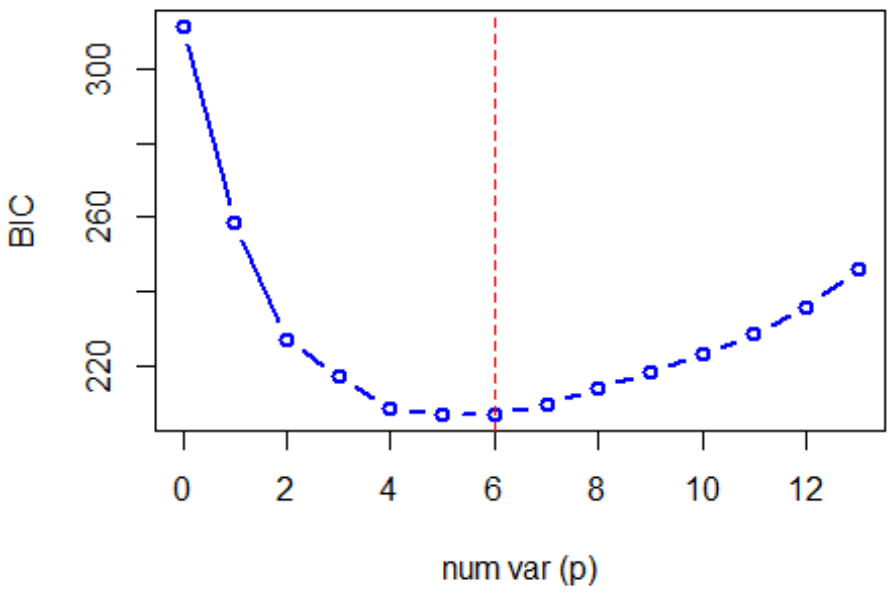
```
# check selected variables in top 5 models
bic.bestglm$BestModels # see which variables are in the selected subset of ea
ch model
```

```
##     age  sex   cp trestbps  chol   fbs restecg thalach exang oldpeak slope
   ca
## 1 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE   FALSE  TRUE   FALSE  TRUE
 TRUE
## 2 FALSE TRUE TRUE    FALSE FALSE FALSE   FALSE   FALSE  TRUE   FALSE  TRUE
 TRUE
## 3 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE   FALSE FALSE   FALSE  TRUE
 TRUE
## 4 FALSE TRUE TRUE    FALSE FALSE FALSE   FALSE   FALSE  TRUE    TRUE  TRUE
 TRUE
## 5 FALSE TRUE TRUE    FALSE FALSE FALSE   FALSE   FALSE FALSE   FALSE  TRUE
 TRUE
##    thal Criterion
## 1 FALSE  206.7565
## 2 FALSE  206.9396
## 3 FALSE  207.0481
## 4 FALSE  207.9683
## 5 FALSE  208.3392
```

```
##### determine the best p for BIC
bic.reg <- as.numeric(unlist(bic.bestglm$Subsets["BIC"]))
bic.reg # BIC of subset models
```

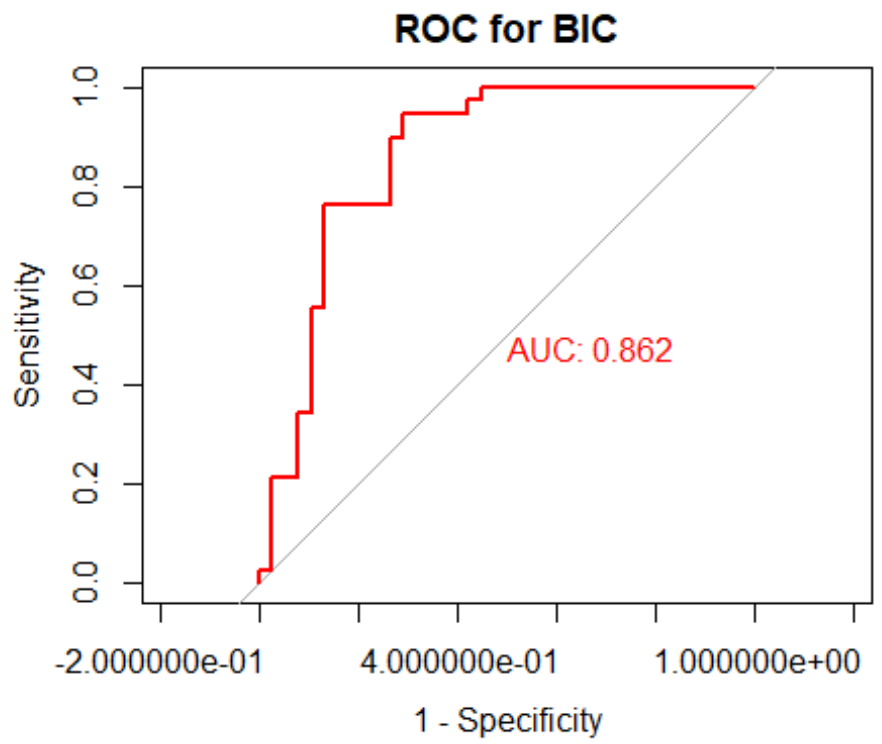
```
##  [1] 311.4698 258.3159 227.0428 216.8434 208.3392 206.9396 206.7565 209.58
77
##  [9] 213.7240 218.3634 222.9634 228.3861 235.5966 245.9735
```

```
p.bicbest <- which.min(bic.reg) - 1 # p for smallest BIC

# plot BIC vs p for each model
plot(x = c(0:13), y = bic.reg, xlab = "num var (p)", ylab = "BIC",
     type = 'b', col = "blue", lwd = 2, cex.lab = 1)
abline(v = p.bicbest, col = "red", lty = 2)
```




```
cat("The best p for subset with smallest BIC is", p.bicbest, "\n")

## The best p for subset with smallest BIC is 6

##### get the selected variables in best model
bic.best.v <- bic.bestglm$BestModels[which.min(bic.bestglm$BestModels$Criteri
on),]
bic.best.v # the best model

##     age  sex   cp trestbps  chol   fbs restecg thalach exang oldpeak slope
   ca
## 1 FALSE TRUE TRUE     TRUE FALSE FALSE   FALSE   FALSE  TRUE   FALSE  TRUE
 TRUE
##    thal Criterion
## 1 FALSE  206.7565

bic.sv <- unique(colnames(bic.best.v)[which(bic.best.v == "TRUE", arr.ind = T
RUE)[,2]])
cat("The selected variables for best model with BIC are:", bic.sv, "\n")

## The selected variables for best model with BIC are: sex cp trestbps exang
slope ca

##### fit the selected variables on train set
bic.fit <- glm(y ~ sex + cp + trestbps + exang + slope + ca,
               data = train2, family = binomial)
summary(bic.fit)
```

```
##
## Call:
## glm(formula = y ~ sex + cp + trestbps + exang + slope + ca, family = binom
ial,
##     data = train2)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.55365  -0.39074   0.08826   0.42329   3.03201
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    3.1796     1.1532   2.757 0.005832 **
## sex1          -2.5277     0.5736  -4.407 1.05e-05 ***
## cp1            0.9723     0.6365   1.528 0.126633
## cp2            2.1214     0.5740   3.695 0.000219 ***
## cp3            2.2133     0.8633   2.564 0.010356 *
## trestbps      -3.1269     1.3617  -2.296 0.021661 *
## exang1        -1.1927     0.4998  -2.386 0.017022 *
## slope1        -0.3575     0.7691  -0.465 0.642028
## slope2         1.8052     0.7891   2.288 0.022162 *
## ca1           -2.6850     0.5931  -4.527 5.97e-06 ***
## ca2           -3.7607     0.8174  -4.601 4.21e-06 ***
## ca3           -3.8510     1.2547  -3.069 0.002145 **
## ca4           14.9831  1232.4932   0.012 0.990301
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 311.47  on 226  degrees of freedom
## Residual deviance: 141.66  on 214  degrees of freedom
## AIC: 167.66
##
## Number of Fisher Scoring iterations: 15

##### predict on test set
pi.bic <- predict(bic.fit, newdata = test2, type = "response")

# plot ROC curve and AUC
roc.bic <- roc(y ~ pi.bic, data = test2, plot = TRUE,
               legacy.axes = TRUE, col = "red", lwd = 2,
               main = "ROC for BIC",
               print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

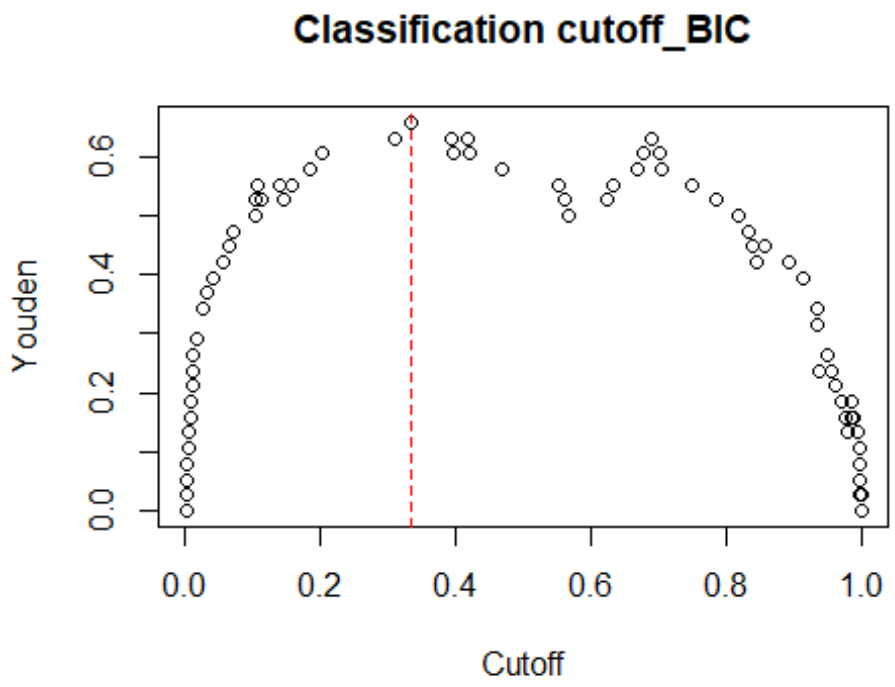



```
# find the optimal cutoff for classification
roc.bic.y <- data.frame(pi.bic, observed.class = test2$y)
bic.opt <- optimal.cutpoints(X = "pi.bic",
                             status = "observed.class",
                             tag.healthy = 0,
                             methods = "Youden",
                             data = roc.bic.y)
# other methods include "MaxEfficiency" for max accuracy, similar to min erro
r rate
bic.opt # optimal cutoff

##
## Call:
## optimal.cutpoints.default(X = "pi.bic", status = "observed.class",
##     tag.healthy = 0, methods = "Youden", data = roc.bic.y)
##
## Optimal cutoffs:
##   Youden
## 1 0.3355
##
## Area under the ROC curve (AUC):  0.862 (0.774, 0.95)

# find sensitivity and specificity under optimal cutoff
bic.tmp <- bic.opt$Youden$Global$measures.acc
bic.youden <- data.frame(bic.tmp$cutoffs, bic.tmp$Se,
                             bic.tmp$Sp,
                             bic.tmp$Se + bic.tmp$Sp - 1)
```

```r
colnames(bic.youden) <- c("Cutoffs", "Sensitivity",
                          "Specificity", "Youden")
bic.youden # all possible cutoffs
```

```
##        Cutoffs Sensitivity Specificity     Youden
## 1  0.001233907  1.00000000  0.00000000 0.00000000
## 2  0.002100354  1.00000000  0.02631579 0.02631579
## 3  0.002223716  1.00000000  0.05263158 0.05263158
## 4  0.003948855  1.00000000  0.07894737 0.07894737
## 5  0.004131137  1.00000000  0.10526316 0.10526316
## 6  0.006889499  1.00000000  0.13157895 0.13157895
## 7  0.007529351  1.00000000  0.15789474 0.15789474
## 8  0.009860764  1.00000000  0.18421053 0.18421053
## 9  0.011009808  1.00000000  0.21052632 0.21052632
## 10 0.011335691  1.00000000  0.23684211 0.23684211
## 11 0.012359939  1.00000000  0.26315789 0.26315789
## 12 0.017033241  1.00000000  0.28947368 0.28947368
## 13 0.026592133  1.00000000  0.34210526 0.34210526
## 14 0.032014831  1.00000000  0.36842105 0.36842105
## 15 0.040789009  1.00000000  0.39473684 0.39473684
## 16 0.057742181  1.00000000  0.42105263 0.42105263
## 17 0.064109969  1.00000000  0.44736842 0.44736842
## 18 0.069745925  1.00000000  0.47368421 0.47368421
## 19 0.104651187  1.00000000  0.50000000 0.50000000
## 20 0.105517215  1.00000000  0.52631579 0.52631579
## 21 0.106750480  1.00000000  0.55263158 0.55263158
## 22 0.113180820  0.97368421  0.55263158 0.52631579
## 23 0.141011201  0.97368421  0.57894737 0.55263158
## 24 0.144845341  0.94736842  0.57894737 0.52631579
## 25 0.159040732  0.94736842  0.60526316 0.55263158
## 26 0.186125137  0.94736842  0.63157895 0.57894737
## 27 0.202556848  0.94736842  0.65789474 0.60526316
## 28 0.309950970  0.94736842  0.68421053 0.63157895
## 29 0.335527272  0.94736842  0.71052632 0.65789474
## 30 0.394363697  0.92105263  0.71052632 0.63157895
## 31 0.398029743  0.89473684  0.71052632 0.60526316
## 32 0.417135067  0.89473684  0.73684211 0.63157895
## 33 0.419413028  0.86842105  0.73684211 0.60526316
## 34 0.469575895  0.84210526  0.73684211 0.57894737
## 35 0.550982537  0.81578947  0.73684211 0.55263158
## 36 0.560066282  0.78947368  0.73684211 0.52631579
## 37 0.567675290  0.76315789  0.73684211 0.50000000
## 38 0.624423574  0.76315789  0.76315789 0.52631579
## 39 0.633651805  0.76315789  0.78947368 0.55263158
## 40 0.669228630  0.76315789  0.81578947 0.57894737
## 41 0.678327323  0.76315789  0.84210526 0.60526316
## 42 0.691063088  0.76315789  0.86842105 0.63157895
## 43 0.702069254  0.73684211  0.86842105 0.60526316
## 44 0.704771234  0.71052632  0.86842105 0.57894737
## 45 0.750276396  0.68421053  0.86842105 0.55263158
```

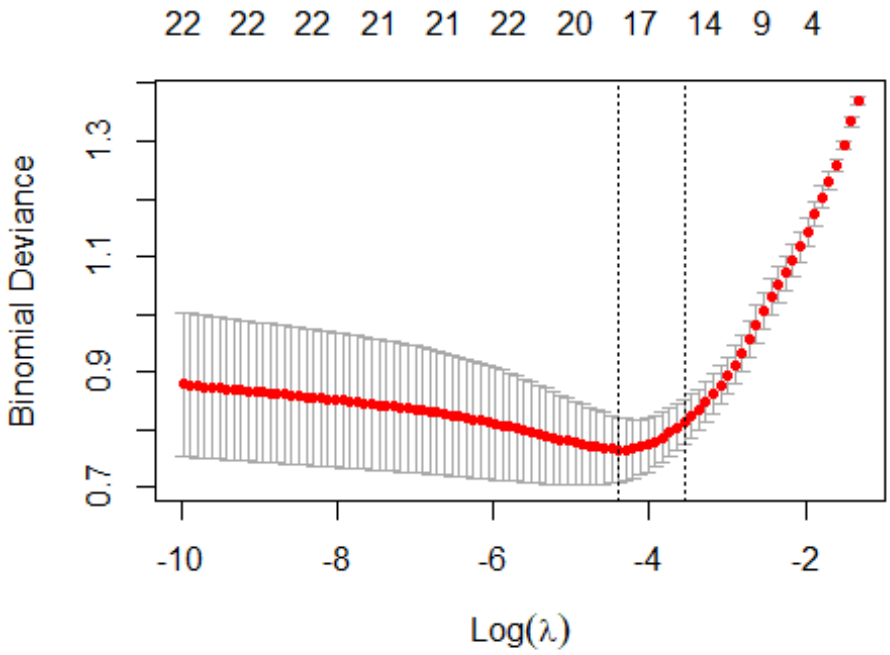
```
## 46 0.785296149  0.65789474  0.86842105 0.52631579
## 47 0.817750285  0.63157895  0.86842105 0.50000000
## 48 0.832076018  0.60526316  0.86842105 0.47368421
## 49 0.838711982  0.57894737  0.86842105 0.44736842
## 50 0.844233730  0.55263158  0.86842105 0.42105263
## 51 0.858102300  0.55263158  0.89473684 0.44736842
## 52 0.892897187  0.52631579  0.89473684 0.42105263
## 53 0.914306901  0.50000000  0.89473684 0.39473684
## 54 0.933705075  0.44736842  0.89473684 0.34210526
## 55 0.934770359  0.42105263  0.89473684 0.31578947
## 56 0.938393349  0.34210526  0.89473684 0.23684211
## 57 0.949138553  0.34210526  0.92105263 0.26315789
## 58 0.955390159  0.31578947  0.92105263 0.23684211
## 59 0.961633270  0.28947368  0.92105263 0.21052632
## 60 0.971966870  0.26315789  0.92105263 0.18421053
## 61 0.975560866  0.23684211  0.92105263 0.15789474
## 62 0.980598661  0.21052632  0.92105263 0.13157895
## 63 0.984714090  0.21052632  0.94736842 0.15789474
## 64 0.986925276  0.21052632  0.97368421 0.18421053
## 65 0.988698008  0.18421053  0.97368421 0.15789474
## 66 0.994775338  0.15789474  0.97368421 0.13157895
## 67 0.997258980  0.13157895  0.97368421 0.10526316
## 68 0.997769230  0.10526316  0.97368421 0.07894737
## 69 0.998237345  0.07894737  0.97368421 0.05263158
## 70 0.998762175  0.05263158  0.97368421 0.02631579
## 71 0.999997656  0.02631579  0.97368421 0.00000000
## 72 0.999999988  0.02631579  1.00000000 0.02631579

# plot cutoffs vs Youden
plot(bic.youden$Cutoffs, bic.youden$Youden,
     xlab = "Cutoff", ylab = "Youden",
     main = "Classification cutoff_BIC")
abline(v = bic.opt[[1]]$Global$optimal.cutoff$cutoff,
       col = "red", lty = 2) # cutoffs for max Youden
```



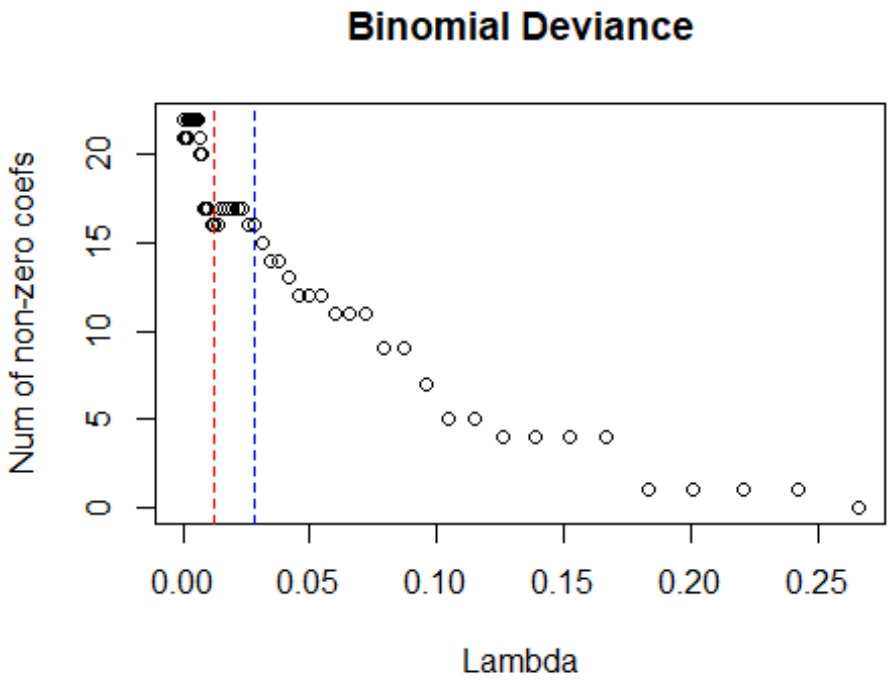

```
bic.youden.opt <- bic.youden[which.max(bic.youden$Youden), ]
bic.youden.opt # sens and spec under optimal cutoff

##      Cutoffs Sensitivity Specificity    Youden
## 29 0.3355273   0.9473684   0.7105263 0.6578947
```

1.4. LASSO

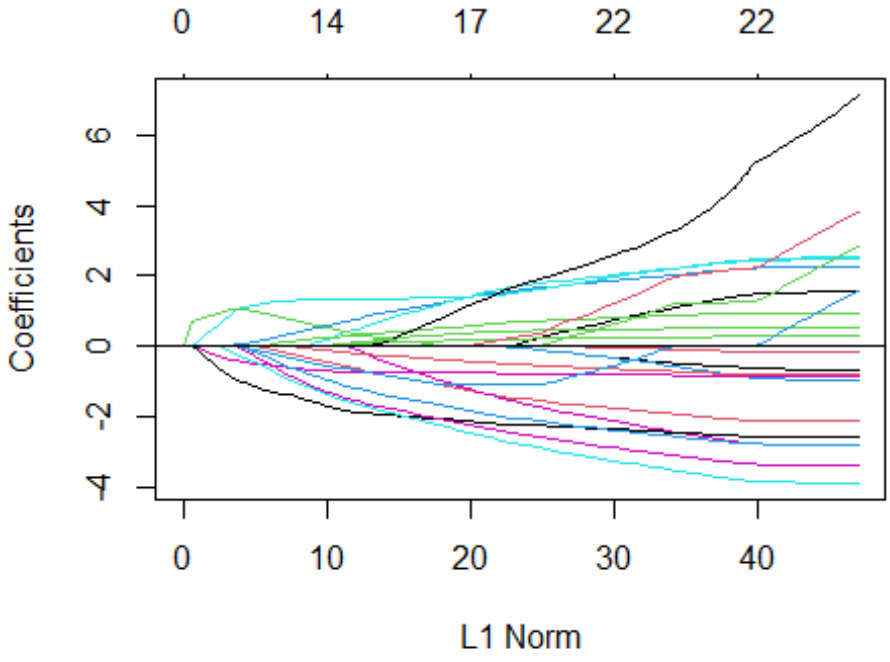
```
# dummy code categorical predictor variables
x.train <- model.matrix(target ~ ., train)[, -1] # no intercept
x.test <- model.matrix(target ~ ., test)[, -1] # no intercept

# convert the outcome (class) to a numerical variable
y.train <- ifelse(train$target == "1", 1, 0)
y.test <- ifelse(test$target == "1", 1, 0)

##### find the best lambda using cross-validation
set.seed(123)
cv.lasso <- cv.glmnet(x.train, y.train, alpha = 1, # LASSO
                      family = "binomial",
                      type.measure = "deviance",
                      nfold = 10)
# other type.measure include "class", "auc" for logistic reg
# and "mse" for linear

# plot binomial deviance vs lambda
plot(cv.lasso)
```




```
##### get the good lambda values
las.lmin <- cv.lasso$lambda.min
las.l1se <- cv.lasso$lambda.1se
cat("The lambda min and lambda 1se for LASSO are:",
    las.lmin, las.l1se, "\n")

## The lambda min and lambda 1se for LASSO are: 0.01233446 0.0284942

# plot number of non-zero coefficients vs lambda
plot(cv.lasso$nzero ~ cv.lasso$lambda,
     xlab = "Lambda", ylab = "Num of non-zero coefs",
     main = cv.lasso$name)
abline(v = c(las.lmin, las.l1se), col = c("red", "blue"), lty = 2)
```




```r
# plot coefficient shrinkage vs lambda
plot(cv.lasso$glmnet.fit)
abline(h = 0)
```



```r
# get the 1se coefficients
bhatL1 <- coef(cv.lasso$glmnet.fit, s = las.l1se)[, 1] # [,1] gets rid of spa
rse matrix format
names(bhatL1[abs(bhatL1) == 0]) # which ones are 0
```

```
## [1] "age"      "chol"     "fbs1"     "restecg2" "ca4"      "thal1"
```

```r
names(bhatL1[abs(bhatL1) != 0]) # which ones are selected
```

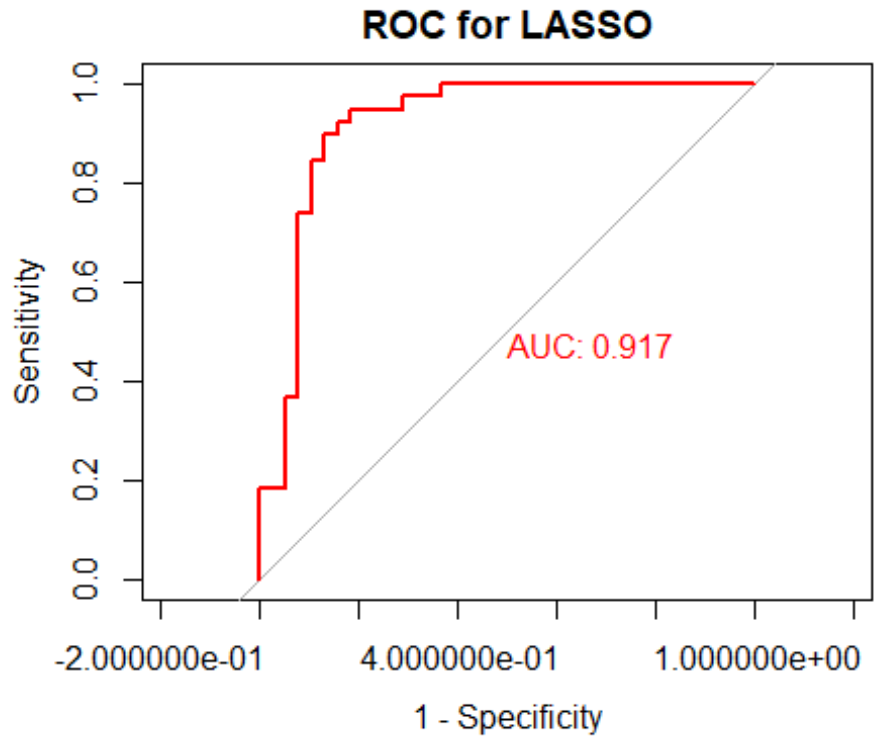
```
##  [1] "(Intercept)" "sex1"        "cp1"         "cp2"         "cp3"
##  [6] "trestbps"    "restecg1"    "thalach"     "exang1"      "oldpeak"
## [11] "slope1"      "slope2"      "ca1"         "ca2"         "ca3"
## [16] "thal2"       "thal3"
```

```r
##### fit the final model on train set
las.fit <- glmnet(x.train, y.train, alpha = 1,
                  family = "binomial",
                  lambda = las.l1se) # use las.1lse for more zero coefs

# display regression coefficients
coef(las.fit)
```

```
## 23 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  0.77054046
## age          .
## sex1        -0.65539411
## cp1          0.09036520
## cp2          0.77235831
## cp3          0.49059262
## trestbps    -0.11127160
## chol         .
## fbs1         .
## restecg1     0.07361214
## restecg2     .
## thalach      1.32769714
## exang1      -0.72895376
## oldpeak     -1.87643014
## slope1      -0.21171447
## slope2       0.30954953
## ca1         -1.19875872
## ca2         -1.65182578
## ca3         -1.56319639
## ca4          .
## thal1        .
## thal2        0.36979982
## thal3       -0.70843413
```

```r
##### predict on test set
pi.las <- predict(las.fit, newx = x.test, s = las.l1se,
                  type = "response")
```

```
pi.las <- as.numeric(pi.las)

# plot ROC curve and AUC
roc.las <- roc(target ~ pi.las, data = test,
               plot = TRUE, legacy.axes = TRUE,
               col = "red", lwd = 2,
               main = "ROC for LASSO",
               print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
# find the optimal cutoff for classification
roc.las.y <- data.frame(pi.las, observed.class = test$target)
las.opt <- optimal.cutpoints(X = "pi.las",
                             status = "observed.class",
                             tag.healthy = 0,
                             methods = "Youden",
                             data = roc.las.y)
# other methods include "MaxEfficiency" for max accuracy, similar to min erro
r rate
las.opt # optimal cutoff

##
## Call:
## optimal.cutpoints.default(X = "pi.las", status = "observed.class",
```

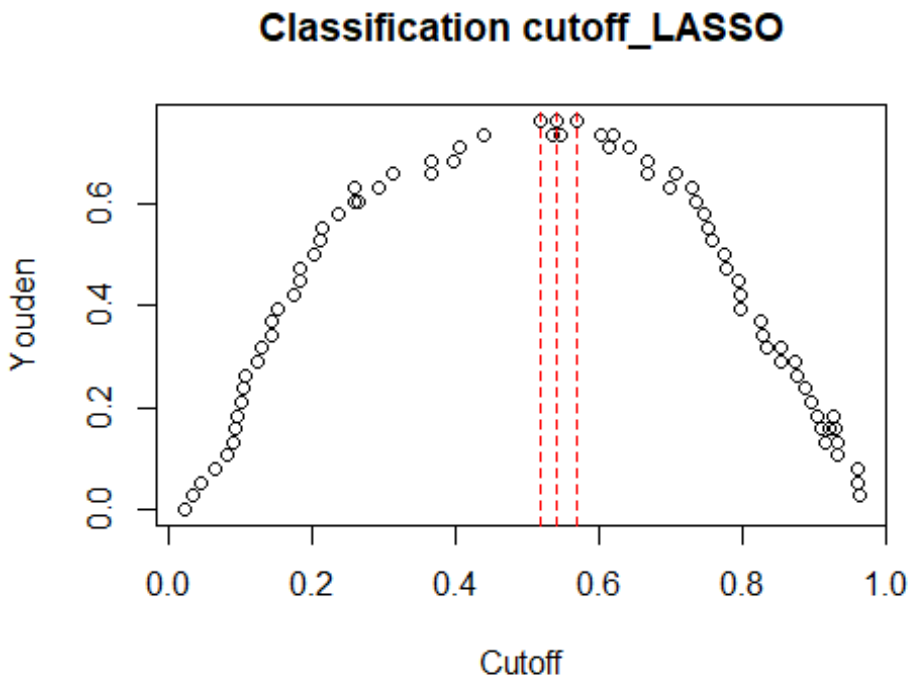
```
##     tag.healthy = 0, methods = "Youden", data = roc.las.y)
##
## Optimal cutoffs:
##   Youden
## 1 0.5184
## 2 0.5409
## 3 0.5690
##
## Area under the ROC curve (AUC):  0.917 (0.847, 0.987)
```

```
# find sensitivity and specificity under optimal cutoff
las.tmp <- las.opt$Youden$Global$measures.acc
las.youden <- data.frame(las.tmp$cutoffs, las.tmp$Se,
                         las.tmp$Sp,
                         las.tmp$Se + las.tmp$Sp - 1)
colnames(las.youden) <- c("Cutoffs", "Sensitivity",
                          "Specificity", "Youden")
las.youden # all possible cutoffs
```

```
##       Cutoffs Sensitivity Specificity     Youden
## 1  0.02133024  1.00000000  0.00000000 0.00000000
## 2  0.03333814  1.00000000  0.02631579 0.02631579
## 3  0.04543803  1.00000000  0.05263158 0.05263158
## 4  0.06602962  1.00000000  0.07894737 0.07894737
## 5  0.08179738  1.00000000  0.10526316 0.10526316
## 6  0.08893935  1.00000000  0.13157895 0.13157895
## 7  0.09161110  1.00000000  0.15789474 0.15789474
## 8  0.09596244  1.00000000  0.18421053 0.18421053
## 9  0.10273694  1.00000000  0.21052632 0.21052632
## 10 0.10517741  1.00000000  0.23684211 0.23684211
## 11 0.10713544  1.00000000  0.26315789 0.26315789
## 12 0.12395727  1.00000000  0.28947368 0.28947368
## 13 0.13068629  1.00000000  0.31578947 0.31578947
## 14 0.14319664  1.00000000  0.34210526 0.34210526
## 15 0.14486080  1.00000000  0.36842105 0.36842105
## 16 0.15183989  1.00000000  0.39473684 0.39473684
## 17 0.17597498  1.00000000  0.42105263 0.42105263
## 18 0.18278716  1.00000000  0.44736842 0.44736842
## 19 0.18402740  1.00000000  0.47368421 0.47368421
## 20 0.20198943  1.00000000  0.50000000 0.50000000
## 21 0.21242983  1.00000000  0.52631579 0.52631579
## 22 0.21479208  1.00000000  0.55263158 0.55263158
## 23 0.23728531  1.00000000  0.57894737 0.57894737
## 24 0.25841477  1.00000000  0.60526316 0.60526316
## 25 0.25960613  1.00000000  0.63157895 0.63157895
## 26 0.26461483  0.97368421  0.63157895 0.60526316
## 27 0.29434287  0.97368421  0.65789474 0.63157895
## 28 0.31142332  0.97368421  0.68421053 0.65789474
## 29 0.36549771  0.97368421  0.71052632 0.68421053
## 30 0.36731275  0.94736842  0.71052632 0.65789474
```

```
## 31 0.39641091  0.94736842  0.73684211 0.68421053
## 32 0.40450169  0.94736842  0.76315789 0.71052632
## 33 0.44071667  0.94736842  0.78947368 0.73684211
## 34 0.51839765  0.94736842  0.81578947 0.76315789
## 35 0.53471454  0.92105263  0.81578947 0.73684211
## 36 0.54090089  0.92105263  0.84210526 0.76315789
## 37 0.54654249  0.89473684  0.84210526 0.73684211
## 38 0.56904061  0.89473684  0.86842105 0.76315789
## 39 0.60322285  0.86842105  0.86842105 0.73684211
## 40 0.61420531  0.84210526  0.86842105 0.71052632
## 41 0.61960593  0.84210526  0.89473684 0.73684211
## 42 0.64319765  0.81578947  0.89473684 0.71052632
## 43 0.66768418  0.78947368  0.89473684 0.68421053
## 44 0.66846821  0.76315789  0.89473684 0.65789474
## 45 0.69927537  0.73684211  0.89473684 0.63157895
## 46 0.70821397  0.73684211  0.92105263 0.65789474
## 47 0.73098911  0.71052632  0.92105263 0.63157895
## 48 0.73467817  0.68421053  0.92105263 0.60526316
## 49 0.74582721  0.65789474  0.92105263 0.57894737
## 50 0.75163070  0.63157895  0.92105263 0.55263158
## 51 0.75704880  0.60526316  0.92105263 0.52631579
## 52 0.77570641  0.57894737  0.92105263 0.50000000
## 53 0.77721712  0.55263158  0.92105263 0.47368421
## 54 0.79554238  0.52631579  0.92105263 0.44736842
## 55 0.79719543  0.50000000  0.92105263 0.42105263
## 56 0.79738042  0.47368421  0.92105263 0.39473684
## 57 0.82484211  0.44736842  0.92105263 0.36842105
## 58 0.82848201  0.42105263  0.92105263 0.34210526
## 59 0.83408636  0.39473684  0.92105263 0.31578947
## 60 0.85359762  0.36842105  0.92105263 0.28947368
## 61 0.85471000  0.36842105  0.94736842 0.31578947
## 62 0.87375264  0.34210526  0.94736842 0.28947368
## 63 0.87751001  0.31578947  0.94736842 0.26315789
## 64 0.88657011  0.28947368  0.94736842 0.23684211
## 65 0.89616427  0.26315789  0.94736842 0.21052632
## 66 0.90574052  0.23684211  0.94736842 0.18421053
## 67 0.90935006  0.21052632  0.94736842 0.15789474
## 68 0.91723349  0.18421053  0.94736842 0.13157895
## 69 0.92032070  0.18421053  0.97368421 0.15789474
## 70 0.92663781  0.18421053  1.00000000 0.18421053
## 71 0.93106498  0.15789474  1.00000000 0.15789474
## 72 0.93234562  0.13157895  1.00000000 0.13157895
## 73 0.93264562  0.10526316  1.00000000 0.10526316
## 74 0.96009462  0.07894737  1.00000000 0.07894737
## 75 0.96094152  0.05263158  1.00000000 0.05263158
## 76 0.96345359  0.02631579  1.00000000 0.02631579

# plot cutoffs vs Youden
plot(las.youden$Cutoffs, las.youden$Youden,
     xlab = "Cutoff", ylab = "Youden",
```

```
     main = "Classification cutoff_LASSO")
abline(v = las.opt[[1]]$Global$optimal.cutoff$cutoff,
       col = "red", lty = 2) # cutoffs for max Youden
```

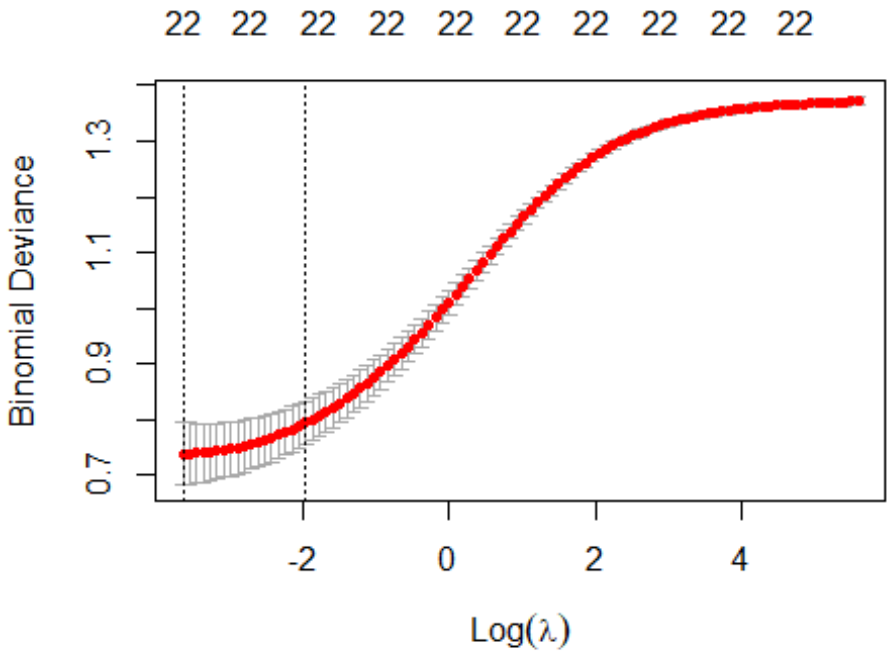



```
las.youden.opt <- las.youden[which(las.youden$Youden
                                   == max(las.youden$Youden)), ]
las.youden.opt # sens and spec under optimal cutoff
```

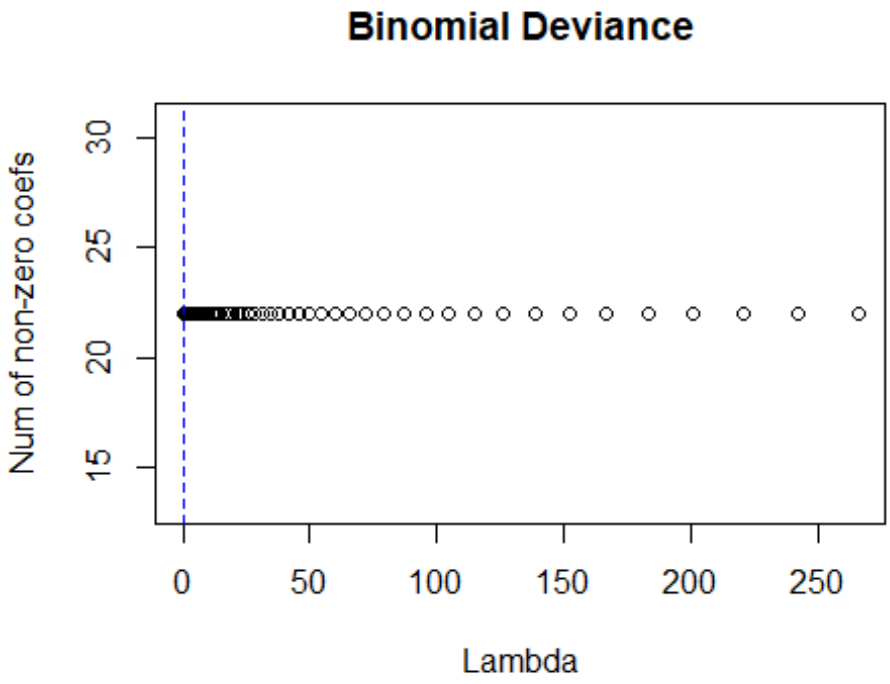
```
##      Cutoffs Sensitivity Specificity    Youden
## 34 0.5183977   0.9473684   0.8157895 0.7631579
## 36 0.5409009   0.9210526   0.8421053 0.7631579
## 38 0.5690406   0.8947368   0.8684211 0.7631579
```

1.5. Ridge

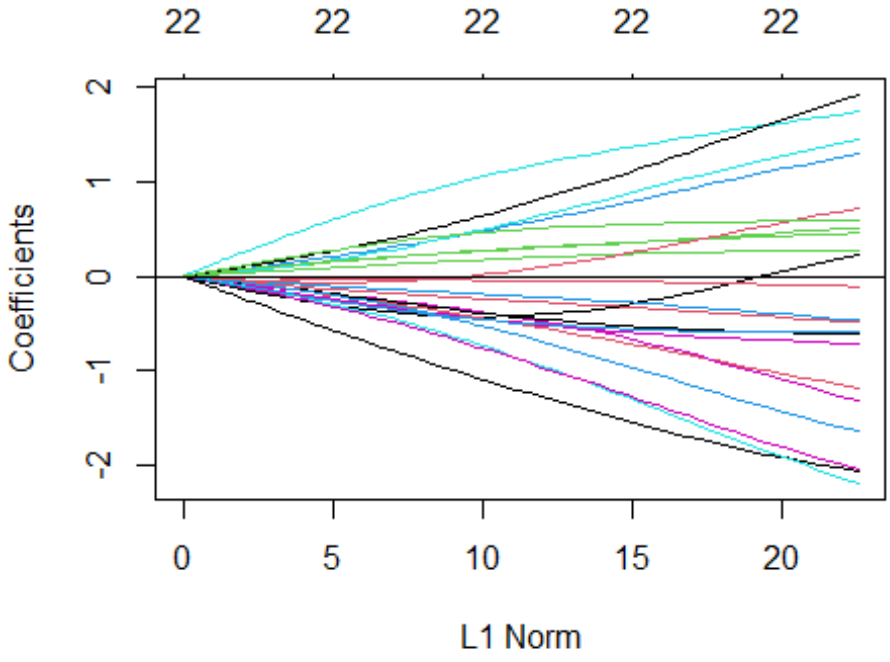
```
##### find the best lambda using cross-validation
set.seed(123)
cv.ridge <- cv.glmnet(x.train, y.train, alpha = 0, # Ridge
                      family = "binomial",
                      type.measure = "deviance",
                      nfold = 10)
# other type.measure include "class", "auc" for logistic reg
# and "mse" for linear

# plot binomial deviance vs lambda
plot(cv.ridge)
```




```
##### get the good lambda values
rid.lmin <- cv.ridge$lambda.min
rid.l1se <- cv.ridge$lambda.1se
cat("The lambda min and lambda 1se for Ridge are:",
    rid.lmin, rid.l1se, "\n")

## The lambda min and lambda 1se for Ridge are: 0.02657379 0.1418163

# plot number of non-zero coefficients vs lambda
plot(cv.ridge$nzero ~ cv.ridge$lambda,
     xlab = "Lambda", ylab = "Num of non-zero coefs",
     main = cv.ridge$name)
abline(v = c(rid.lmin, rid.l1se), col = c("red", "blue"), lty = 2)
```

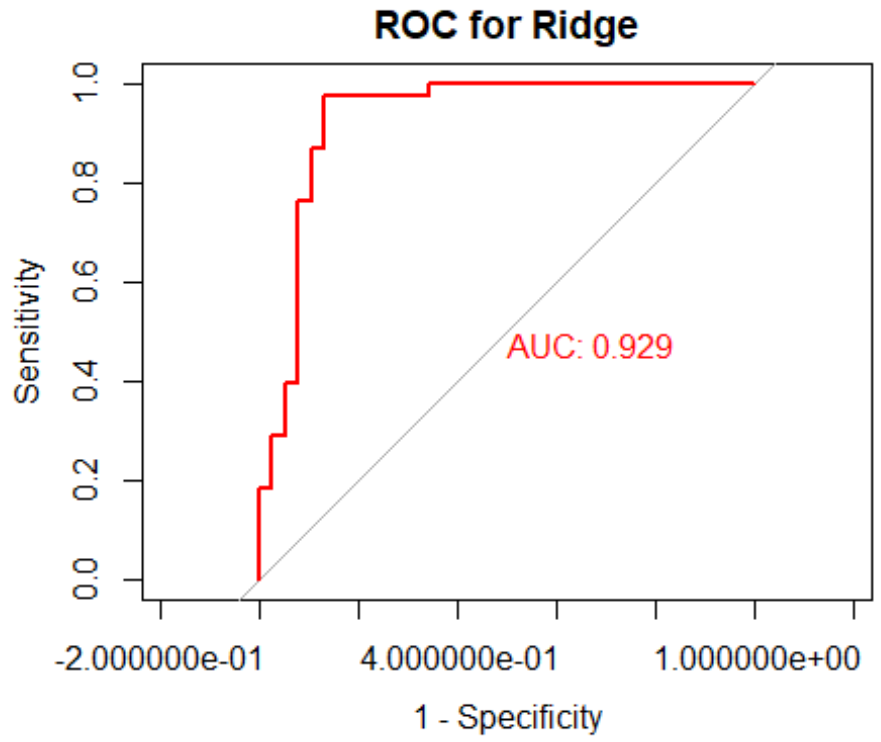



```
# plot coefficient shrinkage vs lambda
plot(cv.ridge$glmnet.fit)
abline(h = 0)
```



```r
# get the 1se coefficients
bhatL0 <- coef(cv.ridge$glmnet.fit, s = rid.l1se)[, 1] # [,1] gets rid of spa
rse matrix format
names(bhatL0[abs(bhatL0) == 0]) # which ones are 0

## character(0)

names(bhatL0[abs(bhatL0) != 0]) # which ones are selected

##  [1] "(Intercept)" "age"         "sex1"        "cp1"         "cp2"
##  [6] "cp3"         "trestbps"    "chol"        "fbs1"        "restecg1"
## [11] "restecg2"    "thalach"     "exang1"      "oldpeak"     "slope1"
## [16] "slope2"      "ca1"         "ca2"         "ca3"         "ca4"
## [21] "thal1"       "thal2"       "thal3"

##### fit the final model on train set
rid.fit <- glmnet(x.train, y.train, alpha = 0,
                  family = "binomial",
                  lambda = rid.l1se) # use rid.1lse for more zero coefs

# display regression coefficients
coef(rid.fit)

## 23 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## (Intercept)  0.70016330
## age         -0.37813015
## sex1        -0.59760090
## cp1          0.32402819
## cp2          0.65184118
## cp3          0.71377466
## trestbps    -0.52461165
## chol        -0.47991164
## fbs1        -0.05000518
## restecg1     0.21249846
## restecg2    -0.23760952
## thalach      1.24615367
## exang1      -0.54262751
## oldpeak     -1.35980588
## slope1      -0.29011766
## slope2       0.31627667
## ca1         -0.77092762
## ca2         -1.05229251
## ca3         -1.05335435
## ca4          0.89947636
## thal1        0.13551235
## thal2        0.52149320
## thal3       -0.52930806

##### predict on test set
pi.rid <- predict(rid.fit, newx = x.test, s = rid.l1se,
```

```
                    type = "response")
pi.rid <- as.numeric(pi.rid)

# plot ROC curve and AUC
roc.rid <- roc(target ~ pi.rid, data = test,
               plot = TRUE, legacy.axes = TRUE,
               col = "red", lwd = 2,
               main = "ROC for Ridge",
               print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
# find the optimal cutoff for classification
roc.rid.y <- data.frame(pi.rid, observed.class = test$target)
rid.opt <- optimal.cutpoints(X = "pi.rid",
                             status = "observed.class",
                             tag.healthy = 0,
                             methods = "Youden",
                             data = roc.rid.y)
# other methods include "MaxEfficiency" for max accuracy, similar to min erro
r rate
rid.opt # optimal cutoff

##
## Call:
```

```
## optimal.cutpoints.default(X = "pi.rid", status = "observed.class",
##     tag.healthy = 0, methods = "Youden", data = roc.rid.y)
##
## Optimal cutoffs:
##   Youden
## 1 0.5039
##
## Area under the ROC curve (AUC):  0.929 (0.863, 0.994)
```

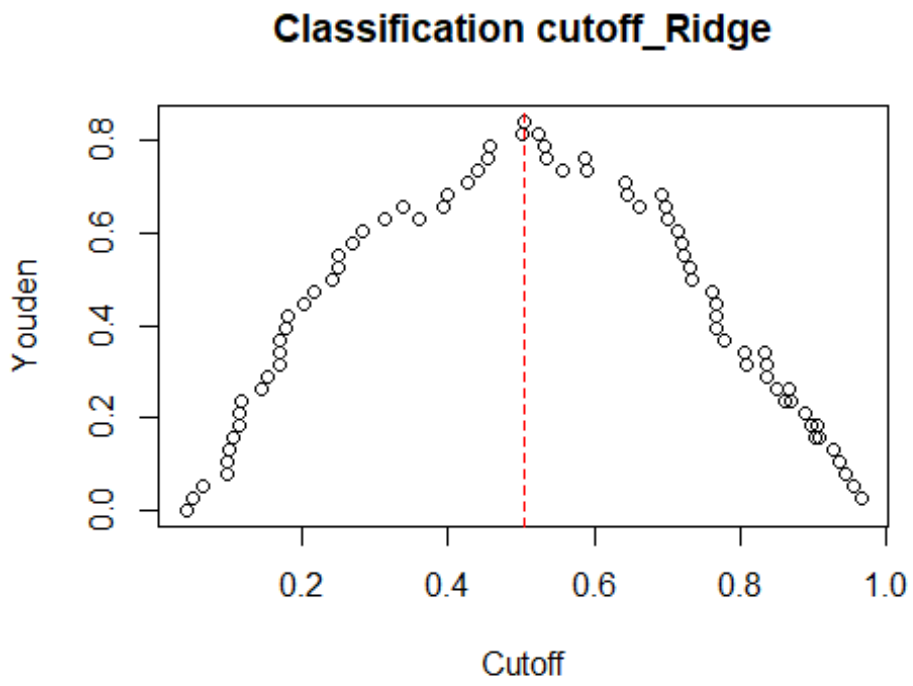
```
# find sensitivity and specificity under optimal cutoff
rid.tmp <- rid.opt$Youden$Global$measures.acc
rid.youden <- data.frame(rid.tmp$cutoffs, rid.tmp$Se,
                         rid.tmp$Sp,
                         rid.tmp$Se + rid.tmp$Sp - 1)
colnames(rid.youden) <- c("Cutoffs", "Sensitivity",
                          "Specificity", "Youden")
rid.youden # all possible cutoffs
```

```
##       Cutoffs Sensitivity Specificity     Youden
## 1  0.04205583  1.00000000  0.00000000 0.00000000
## 2  0.05126910  1.00000000  0.02631579 0.02631579
## 3  0.06597270  1.00000000  0.05263158 0.05263158
## 4  0.09810904  1.00000000  0.07894737 0.07894737
## 5  0.09867753  1.00000000  0.10526316 0.10526316
## 6  0.10011464  1.00000000  0.13157895 0.13157895
## 7  0.10781592  1.00000000  0.15789474 0.15789474
## 8  0.11449987  1.00000000  0.18421053 0.18421053
## 9  0.11502902  1.00000000  0.21052632 0.21052632
## 10 0.11841971  1.00000000  0.23684211 0.23684211
## 11 0.14564460  1.00000000  0.26315789 0.26315789
## 12 0.15501570  1.00000000  0.28947368 0.28947368
## 13 0.16942558  1.00000000  0.31578947 0.31578947
## 14 0.17007878  1.00000000  0.34210526 0.34210526
## 15 0.17131033  1.00000000  0.36842105 0.36842105
## 16 0.17849241  1.00000000  0.39473684 0.39473684
## 17 0.18244643  1.00000000  0.42105263 0.42105263
## 18 0.20287989  1.00000000  0.44736842 0.44736842
## 19 0.21661978  1.00000000  0.47368421 0.47368421
## 20 0.24303887  1.00000000  0.50000000 0.50000000
## 21 0.24987460  1.00000000  0.52631579 0.52631579
## 22 0.25133603  1.00000000  0.55263158 0.55263158
## 23 0.27062880  1.00000000  0.57894737 0.57894737
## 24 0.28463968  1.00000000  0.60526316 0.60526316
## 25 0.31418347  1.00000000  0.63157895 0.63157895
## 26 0.33894091  1.00000000  0.65789474 0.65789474
## 27 0.36058141  0.97368421  0.65789474 0.63157895
## 28 0.39523224  0.97368421  0.68421053 0.65789474
## 29 0.39875909  0.97368421  0.71052632 0.68421053
## 30 0.42594982  0.97368421  0.73684211 0.71052632
## 31 0.44231121  0.97368421  0.76315789 0.73684211
```

```
## 32 0.45562264  0.97368421  0.78947368 0.76315789
## 33 0.45769696  0.97368421  0.81578947 0.78947368
## 34 0.50278111  0.97368421  0.84210526 0.81578947
## 35 0.50387918  0.97368421  0.86842105 0.84210526
## 36 0.52496817  0.94736842  0.86842105 0.81578947
## 37 0.53123970  0.92105263  0.86842105 0.78947368
## 38 0.53620836  0.89473684  0.86842105 0.76315789
## 39 0.55656647  0.86842105  0.86842105 0.73684211
## 40 0.58824592  0.86842105  0.89473684 0.76315789
## 41 0.58943531  0.84210526  0.89473684 0.73684211
## 42 0.64329003  0.81578947  0.89473684 0.71052632
## 43 0.64422784  0.78947368  0.89473684 0.68421053
## 44 0.66115147  0.76315789  0.89473684 0.65789474
## 45 0.69115753  0.76315789  0.92105263 0.68421053
## 46 0.69903640  0.73684211  0.92105263 0.65789474
## 47 0.70181922  0.71052632  0.92105263 0.63157895
## 48 0.71534345  0.68421053  0.92105263 0.60526316
## 49 0.72015887  0.65789474  0.92105263 0.57894737
## 50 0.72319864  0.63157895  0.92105263 0.55263158
## 51 0.73093900  0.60526316  0.92105263 0.52631579
## 52 0.73523196  0.57894737  0.92105263 0.50000000
## 53 0.76221668  0.55263158  0.92105263 0.47368421
## 54 0.76635609  0.52631579  0.92105263 0.44736842
## 55 0.76657074  0.50000000  0.92105263 0.42105263
## 56 0.76723145  0.47368421  0.92105263 0.39473684
## 57 0.77817940  0.44736842  0.92105263 0.36842105
## 58 0.80659344  0.42105263  0.92105263 0.34210526
## 59 0.80903407  0.39473684  0.92105263 0.31578947
## 60 0.83398272  0.39473684  0.94736842 0.34210526
## 61 0.83569432  0.36842105  0.94736842 0.31578947
## 62 0.83629123  0.34210526  0.94736842 0.28947368
## 63 0.85118368  0.31578947  0.94736842 0.26315789
## 64 0.86013275  0.28947368  0.94736842 0.23684211
## 65 0.86522628  0.28947368  0.97368421 0.26315789
## 66 0.86989255  0.26315789  0.97368421 0.23684211
## 67 0.88783052  0.23684211  0.97368421 0.21052632
## 68 0.89695057  0.21052632  0.97368421 0.18421053
## 69 0.90342836  0.18421053  0.97368421 0.15789474
## 70 0.90589245  0.18421053  1.00000000 0.18421053
## 71 0.90683454  0.15789474  1.00000000 0.15789474
## 72 0.92701627  0.13157895  1.00000000 0.13157895
## 73 0.93597187  0.10526316  1.00000000 0.10526316
## 74 0.94384013  0.07894737  1.00000000 0.07894737
## 75 0.95498957  0.05263158  1.00000000 0.05263158
## 76 0.96551324  0.02631579  1.00000000 0.02631579
```

```
# plot cutoffs vs Youden
plot(rid.youden$Cutoffs, rid.youden$Youden,
     xlab = "Cutoff", ylab = "Youden",
     main = "Classification cutoff_Ridge")
```

```
abline(v = rid.opt[[1]]$Global$optimal.cutoff$cutoff,
       col = "red", lty = 2) # cutoffs for max Youden
```

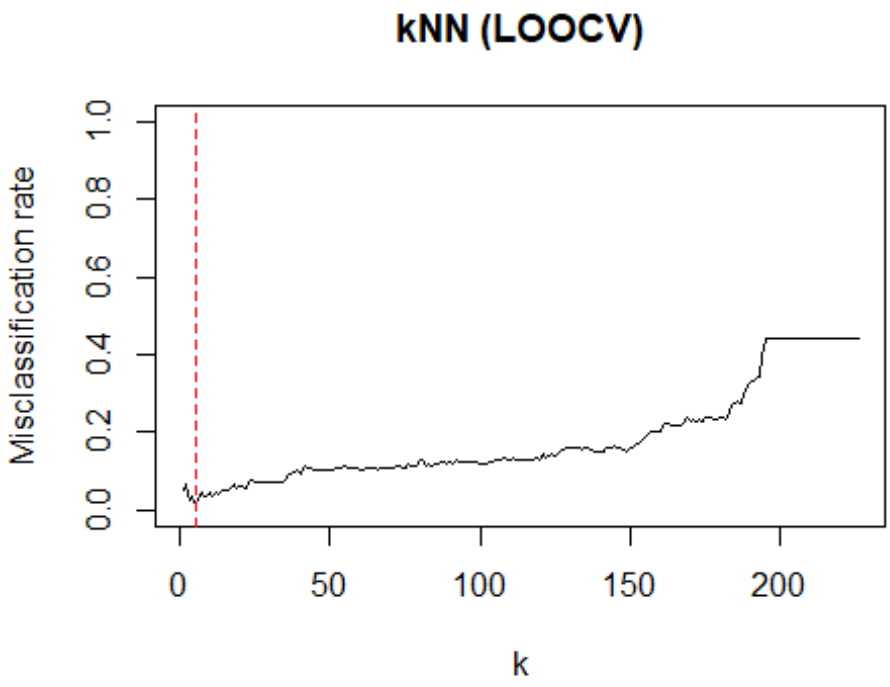



```
rid.youden.opt <- rid.youden[which(rid.youden$Youden
                                   == max(rid.youden$Youden)), ]
rid.youden.opt # sens and spec under optimal cutoff

##      Cutoffs Sensitivity Specificity    Youden
## 35 0.5038792   0.9736842   0.8684211 0.8421053
```

Method 2: kNN

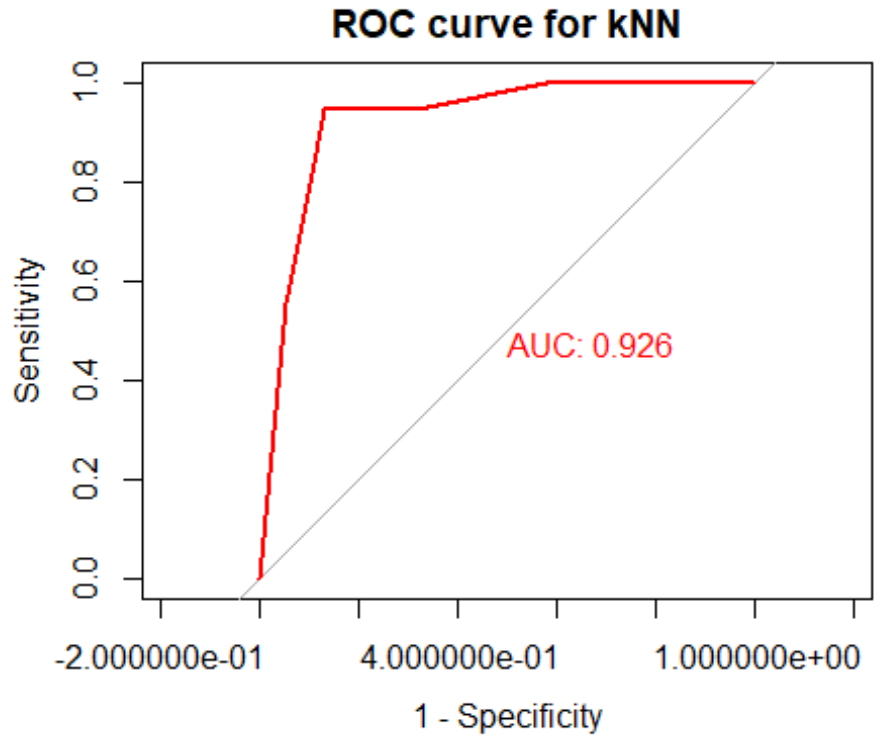
```
##### use leave-one-out cross-validation on train set to choose k
set.seed(123)
k.try <- NULL # store k values tried in loop
mis.knn <- NULL # store misclassification rate for each k tried

for (kv in 1:(length(train$target) - 1)) {
  prob <- knn.cv(train[, -13], train$target, k = kv,
                 prob = TRUE) # LOOCV
  levels(prob) <- levels(target)
  conf.mat <- table(prob, train$target)
  mis <- 1 - sum(diag(conf.mat))/sum(conf.mat)
  mis.knn <- rbind(mis.knn, mis)
  k.try <- rbind(k.try, kv)
}
```

```
##### determine the best k
k.best <- k.try[which.min(mis.knn)] # pick k with smallest misclassifcation r
ate
cat("The best k is", k.best,"\n")

## The best k is 5

cat("The min misclassification rate is", min(mis.knn), ", at k =", k.best, "\
n")

## The min misclassification rate is 0.01762115 , at k = 5

# plot misclassification rate vs k
par(mfrow = c(1, 1))
plot(k.try, mis.knn, type = "l", ylim = c(0, 1), xlab = "k",
     ylab = "Misclassification rate",
     main = "kNN (LOOCV)", cex.lab = 1)
abline(v = k.best, col = "red", lty = 2)
```




```
##### refit k = min misclassification rate on test set
k.fit <- kknn(target ~ ., train, test, k = k.best,
              kernel = "rectangular")

k.pred <- k.fit$fitted.values
levels(k.pred) <- levels(target)
cm.pred.knn <- table(k.pred, test$target)
mis.pred.knn <- 1 - sum(diag(cm.pred.knn))/sum(cm.pred.knn)
```

```
cat("The out-of-sample misclassification rate is", mis.pred.knn,
    "\n")

## The out-of-sample misclassification rate is 0.09210526

# plot ROC curve and AUC on test set
k.prob <- k.fit$prob[, 2]
roc.knn <- roc(response = test$target,
               predictor = factor(k.prob, ordered = TRUE),
               plot = TRUE, legacy.axes = TRUE, col = "red",
               lwd = 2, main = "ROC curve for kNN",
               print.auc = TRUE)

## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```



Random Forest

**** Code ****

```
library(randomForest)
# RF model with default parameters
rf <- randomForest(target ~ ., data=train,)
rf.p1 <- predict(rf,test,type = "prob")
library(pROC)

r1 <- roc(test$target,rf.p1[,2],grid = TRUE,plot=TRUE, print.auc = TRUE)
```

```
# cross validation for mtry = 1:10, ntree = 500
library(caret)
library(e1071)
rf.ctrl <- trainControl(method="repeatedcv", number=5, repeats=3, search="grid")
rf.grid <- expand.grid(.mtry=c(1:10))
rf.tune <- train(target~., data=train, method="rf", tuneGrid=rf.grid, trControl=rf.ctrl)
plot(rf.tune)

# cross validation for mtry = 1:10, ntree = 100, 500, 1000
rf.list <- list()
for (ntree in c(100, 500, 1000)) {
  f <- train(target~., data=train, method="rf", tuneGrid=rf.grid, trControl=rf.ctrl)
  key <- toString(ntree)
  rf.list[[key]] <- f}
rf.list

# selected model: mtry = 1, ntree = 1000
rf.3 <- randomForest(target ~ ., data=train, mtry = 1, ntree = 1000)
rfp3 <- predict(rf.3,test,type = "prob")
r3 <- roc(test$target,rfp3[,2],grid = TRUE,plot=TRUE, print.auc = TRUE)
varImpPlot(rf.3)
```

**** Validation Results ****

Validation Accuracy:

mtry	**Ntree = 100**	**Ntree = 500**	**Ntree = 1000**
1	0.8245	0.8176	**0.8325**
2	0.8298	0.8216	0.8174
3	0.8298	0.8176	0.8215
4	0.8215	0.8093	0.8188
5	0.8202	0.8134	0.8133
6	0.8257	0.8093	0.8119
7	0.8230	0.8066	0.8065
8	0.8189	0.7997	0.8024
9	0.8217	0.8053	0.8024
10	0.8230	0.7971	0.8037

Boosting

**** GBM Code ****

```
library(gbm)
train$target <- as.numeric(train$target) - 1

# gbm tuning grid
b.grid1 <- expand.grid(
  shrinkage = c(.01, 0.05, 0.1),
  interaction.depth = c(1, 2, 3),
  ntree = c(500,1000),
  auc = 0)
nrow(b.grid1) #18

# use 0.25*nrow(train) data for validation
ind <- ceiling(nrow(train)*0.75)
ii <- sample(1:nrow(train),ind)
tr <- train[ii,]
te <- train[-ii,]
for(i in 1:nrow(b.grid1)) {
  m <- gbm(formula = target ~ ., distribution = "bernoulli", data = tr,
           n.trees = b.grid1$ntree[i], interaction.depth = b.grid1$interaction.depth[i],
           shrinkage = b.grid1$shrinkage[i], train.fraction = 1)
  m.yh <- predict(m,te,type = "response")
  b.grid1$auc[i] <- roc(te$target,m.yh)$auc}

b.grid1[which.max(b.grid1$auc),]

# fit selected model on train set
b.t <- gbm(target~.,data =train,
       interaction.depth=2, n.trees=500, shrinkage=0.01,
       distribution = "bernoulli", train.fraction = 1)
bt.yh <- predict(b.t,test,type = "response")
bt.r1 <- roc(test$target,bt.yh,grid = TRUE,plot=TRUE, print.auc = TRUE)
summary(b.t,cBars = 8)
```

**** XGBOOST Code ****

```
library(xgboost)
train_x <- train_n[,1:13]
train_y <- train_n[,14]
test_x <- test_n[,1:13]
test_y <- test_n[,14]

dtrain = xgb.DMatrix(data =  as.matrix(train_x), label = train_y )
dtest = xgb.DMatrix(data =  as.matrix(test_x))
bst = xgb.train(data = dtrain, max.depth = 2, eta = 0.01, nthread = 2, nround = 500,
                objective = "binary:logistic", print_every_n = 500)
bst.yh <- predict(bst,dtest)
bst.roc <- roc(test_y,bst.yh,grid = TRUE,plot=TRUE, print.auc = TRUE) #0.910
```

**** GBM Tuning Results ****

	shrinkage	interaction.depth	ntree	auc
1	0.01	1	500	0.9121244
2	0.05	1	500	0.8854283
3	0.10	1	500	0.8654060
4	0.01	2	500	**0.9121246**
5	0.05	2	500	0.8676307
6	0.10	2	500	0.8476085
7	0.01	3	500	0.9110122
8	0.05	3	500	0.8576196
9	0.10	3	500	0.8453838
10	0.01	1	1000	0.9054505
11	0.05	1	1000	0.8498331
12	0.10	1	1000	0.8409344
13	0.01	2	1000	0.8954394
14	0.05	2	1000	0.8509455
15	0.10	2	1000	0.8286986
16	0.01	3	1000	0.8876529
17	0.05	3	1000	0.8364850
18	0.10	3	1000	0.8342603

Neural Networks

**** See Code in Jupyter Notebook ****