Introduction

The objective of my analyses was to assess the performance of 3 different classification methods (logistic regression with ridge regression regularization, K-nearest neighbours, and classification tree), and to train a model that can be used to classify future observations. I decided to use logistic regression with ridge regression regularization instead of lasso regularization, because like the other 2 methods which it was compared to (K-nearest neighbours, and classification tree), ridge regression does not select coefficients. In this way, the ridge regression method is more comparable to the other classification methods than the lasso method. For the analyses presented here, data from the Cleveland database that is part of a larger collection of databases on heart disease diagnosis was used.

I examined the classification of the presence or absence of heart disease using 13 predictors. The predictor variables sex, chest pain type, fasting blood sugar, resting electrocardiographic results, exercise induced angina, the slope of the peak exercise ST segment, the number of major vessels coloured by fluoroscopy, and thalassemia are categorical. The other predictor variables age, resting blood pressure, serum cholesterol, maximum heart rate achieved, and ST depression induced by exercise relative to rest are numerical. The outcome variable presence or absence of heart disease is categorical, where values less than or equal to 1 represent class 1 (presence of the disease), and values of 0 represent class 0 (absence of the disease).

Methods

Data Preparation

To setup, I loaded the glmnet, pROC, class, epiR, and tree packages to be used. I read the Cleveland dataset from the text file into R using the read.table function specifying that there is no header in the data file, that "," is the field separator character, and to return the character "?" for missing values in the file. I used the na.omit function to create a dataset d with only complete information (without observations with missing values). I encoded the categorical variables (V2, V3, V6, V7, V9, V11, V12, V13, and V14) as factors. I also specified the classes of the outcome variable V14 with values less than or equal to 1 as class 1 (presence of the disease), and values of 0 as class 0 (absence of the disease). Lastly, I named the variables.

Overall Methods for the Assessing the Performance of Different Classification Methods

To assess the performance of models created using the 3 different classification methods, the split set method with 2/3 of the data in the training set was used. Models were trained and tuned using the training dataset. Performance was assessed on the test dataset. Sensitivity and specificity were used to assess the performance of the classification. I decided on this criterion because sensitivity and specificity measure the true positive rate and true negative rate, respectively, and provide separate measures for the misclassification error and accuracy of the 2 classes. As such, sensitivity and specificity provide a thorough assessment of the performance of the classification. The method that produced the models with the highest sensitivity and specificity was used to train a model to be used to classify future observations.

Assess the Performance of a Ridge Regression Model

First, I specified the predictor (V1-13) and outcome (V14) variables in the objects x and y, respectively. Then, I split the Cleveland d dataset into training and test datasets with 2/3 of the data in the training dataset. To tune the model, I used the cv.glmnet function, and the alpha=0 and nfolds=10 arguments to determine the optimal value for lambda that minimizes the MSE of the ridge regression model trained on the training dataset with 10 fold cross-validation.

Using the predict function, I created an object with the predicted classes of the test dataset from the ridge regression model with the optimal lambda value. With the table function, I created a confusion matrix of predicted classes versus true classes. Then, using the epi.tests function, I calculated the sensitivity and specificity. By re-running the R code, I repeated the entire process from splitting the Cleveland d dataset into training and test datasets to calculating the sensitivity and specificity of the model a total of 5 times.

Assess the Performance of a K-Nearest Neighbours Model

First, I specified the predictor (V1-13) and outcome (V14) variables in the objects x and y, respectively, and scaled the predictor variables using the scale function to make things comparable. Then, I split the Cleveland d dataset into training and test datasets with 2/3 of the data in the training dataset. To tune the model, I used the knn function to create an object with the predicted classes of the test dataset from the K-nearest neighbors model trained on the training dataset for multiple values of K (1, 5, and 10). Using the table function, I created a confusion matrix of predicted classes versus true classes. Then, using the epi.tests function, I calculated the sensitivity and specificity. The optimal K value that produced the model with the highest sensitivity and specificity was used to train a K-nearest neighbours model to compare with the other methods.

Using the knn function, I created an object with the predicted classes of the test dataset from the K-nearest neighbors model trained on the training dataset with the optimal K value. Using the table function, I created a confusion matrix of predicted classes versus true classes. Then, using the epi.tests function, I calculated the sensitivity and specificity. By re-running the R code, I repeated the entire process from splitting the Cleveland d dataset into training and test datasets to calculating the sensitivity and specificity of the model a total of 5 times.

Assess the Performance of a Classification Tree Model

First, I specified the predictor (V1-13) and outcome (V14) variables in the objects x and y, respectively. Then, I split the Cleveland d dataset into training and test datasets with 2/3 of the data in the training dataset. Using the tree function, I trained a classification tree using the training dataset. To tune the model, I used the cv.tree function, and the FUN=prune.tree, method="misclass", and K=5 arguments to determine the size (number of terminal nodes) that minimizes the misclassification error using 5-fold cross-validation. Then, I used the prune.misclass function to prune the classification tree to the size (number of terminal nodes) that minimizes the misclassification error.

Using the predict function, I created an object with the predicted classes of the test dataset from the classification tree model with the size (number of terminal nodes) that minimizes the misclassification error. Using the table function, I created a confusion matrix of predicted classes versus true classes. Then, using the epi.tests function, I calculated the sensitivity and specificity. By re-running the R code, I repeated the entire process from splitting the Cleveland d dataset into training and test datasets to calculating the sensitivity and specificity of the model a total of 5 times.

Train a Model to Be Used for Classification of Future Observations

Of the 3 different classification methods (logistic regression with ridge regression regularization, K-nearest neighbours, and classification tree), the method that produces the models that performed best with the highest sensitivity and specificity was used to train the final model that will be used to classify future observations. The final model was trained and tuned on the Cleveland d dataset. Since the classification tree models performed the best, the final model was a classification tree model.

To train the final model, I first specified the predictor (V1-13) and outcome (V14) variables in the objects x and y, respectively. Using the tree function, I trained a classification tree using the Cleveland d dataset. To tune the model, I used the cv.tree function, and the FUN=prune.tree, method="misclass", and K=5 arguments to determine the size (number of terminal nodes) that minimizes the misclassification error using 5-fold cross-validation. Then, I used the prune.misclass function to prune the classification tree to the size (number of terminal nodes) that minimizes the misclassification error. Lastly, I plotted the pruned decision tree.

Results

Assess the Performance of a Ridge Regression Model

Logistic regression models with ridge regression regularization were trained and tuned on the training set then assessed on the test dataset. The optimal lambda values that minimize the MSE, and the sensitivity and specificity of the ridge regression models with the optimal lambda value across 5 repetitions are listed in Table 1.

Table 1. The optimal lambda values, and the sensitivity and specificity of the ridge regression models with the optimal lambda value across 5 trials.

Trial #	Optimal Lambda Value	Sensitivity	Specificity
1	0.03839479	0.15	0.19
2	0.0322885	0.08	0.26
3	0.04334681	0.13	0.21

4	0.04483221	0.10	0.29
5	0.03916323	0.15	0.17

Assess the Performance of a K-Nearest Neighbours Model

K-nearest neighbours models were trained and tuned on the training set then assessed on the test dataset. The optimal K values that produce models with the best performance, and the sensitivity and specificity of the K-nearest neighbours model with the optimal K value across 5 repetitions are listed in Table 2.

Table 2. The optimal K values, and the sensitivity and specificity for the K-nearest neighbours models with the optimal K value across 5 trials.

Trial #	Optimal K Value	Sensitivity	Specificity
1	10	0.58	0.55
2	10	0.81	0.50
3	10	0.65	0.61
4	10	0.61	0.62
5	10	0.72	0.61

Assess the Performance of a Classification Tree Model

Classification tree models were trained and pruned on the training set then assessed on the test dataset. The optimal sizes (number of terminal nodes) that minimize the misclassification error, and the sensitivity and specificity of the pruned classification tree models across 5 repetitions are listed in Table 3.

Table 3. The optimal sizes, and the sensitivity and specificity of the classification tree models with the optimal size across 5 trials.

Trial #	Optimal Size	Sensitivity	Specificity
1	6	0.79	0.62
2	9	0.90	0.70
3	7	0.89	0.74
4	6	0.84	0.80
5	7	0.76	0.78

Train a Model to Be Used for Classification of Future Observations

Of the models created using the 3 different classification methods (logistic regression with ridge regression regularization, K-nearest neighbours, and classification tree), the classification tree models performed the best and had the highest sensitivity and specificity across all 5 repeats of the assessment. As such, a classification tree model was trained to be used to classify future observations.

A classification tree model was trained and tuned on the Cleveland dataset to be used to classify future observations. The size (number of terminal nodes) that minimizes the misclassification error is 10. The pruned classification tree is presented in Figure 1. The final pruned classification tree can be used to classify future observations.

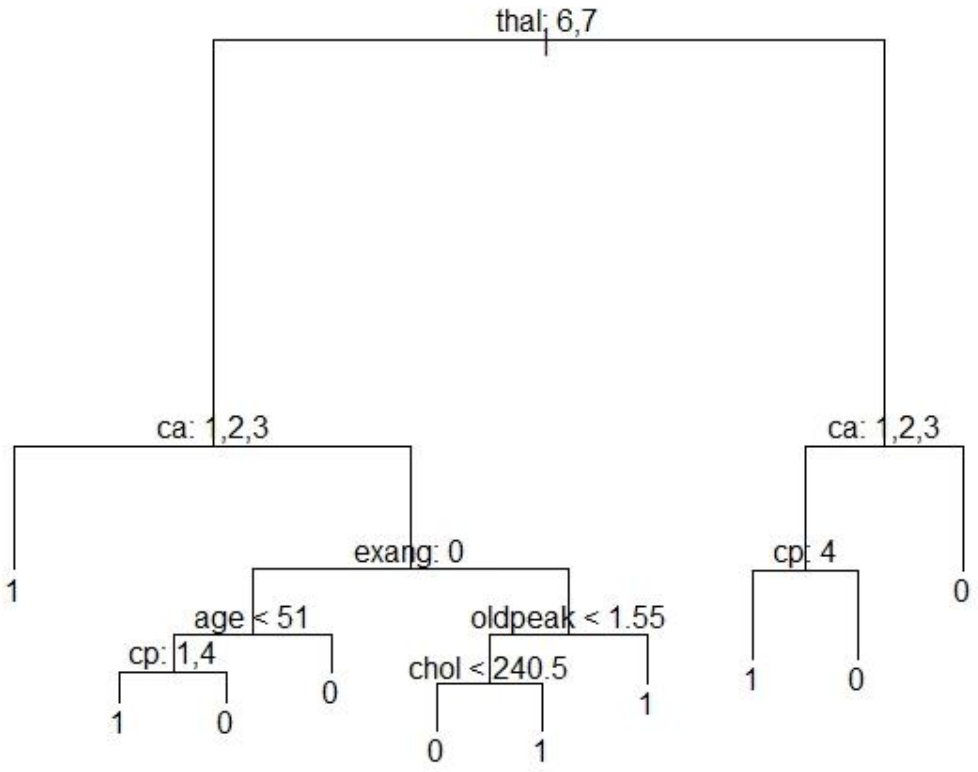

Figure 1. The pruned classification tree to be used to classify future observations.

Discussion and Conclusions

The objective of my analyses was to assess the performance of 3 different classification methods (logistic regression with ridge regression regularization, K-nearest neighbours, and classification tree), and to train a model that can be used to classify future observations. I decided to use logistic regression with ridge regression regularization instead of lasso regularization, because the ridge regression method does not select coefficients and is more comparable to the other classification methods in that way.

To assess the performance of models created using the different classification methods, the split set method with 2/3 of the data in the training set was used. Models were trained and tuned using the training dataset. Performance was assessed on the test dataset. Sensitivity and specificity were used to assess the performance of the classification because they measure the true positive rate and true negative rate, respectively, and provide separate measures for the misclassification error and accuracy of the 2 classes. The method that produced the models with the highest sensitivity and specificity was used to train a model to be used to classify future observations.

If false negatives cost much more than false positives in this classification problem, when training the classifier model, we can decrease the threshold to assign a positive classification. As a prediction is classified as being positive when the probability is above the threshold, decreasing the threshold will increase the sensitivity or true positive rate and lower the false negative rate. Furthermore, when evaluating and using classifier models, we can choose the method and/or model that has a higher sensitivity. When the sensitivity or true positive rate is high, the false negative rate is low. This is true regardless of the classification method used.

For these analyses, for each classification method, I repeated the entire process from splitting the Cleveland dataset into training and test datasets to calculating the sensitivity and specificity of the model 5 times. Of the models created using the 3 different classification methods, the classification tree models performed the best and had the highest sensitivity and specificity across all 5 trials. As such, all 5 repeats of the assessment came to a consensus regarding the best method for this classification problem. Since the classification tree models performed the best, a classification tree model was trained and tuned using the Cleveland dataset to be used to classify future observations.

.

Ivy Fong

Appendix: R Code

```
#load packages to be used
library(glmnet)
library(pROC)
library(class)
library(epiR)
library(tree)

#load data and create dataset d without missing values and where V14<=1 is cl
ass 1, V14=0 is class 0
d <- read.table("cleveland.txt", header=F, sep=",", na.strings="?") #read txt
data into R, specify no variable names, field separator character = ",", retu
rn ? for missing values
d <- na.omit(d) #only keep observations with complete information

#encode categorical variables as factors
d$V2 <- as.factor(d$V2) #encode V2 as a factor
d$V3 <- as.factor(d$V3) #encode V3 as a factor
d$V6 <- as.factor(d$V6) #encode V6 as a factor
d$V7 <- as.factor(d$V7) #encode V7 as a factor
d$V9 <- as.factor(d$V9) #encode V9 as a factor
d$V11 <- as.factor(d$V11) #encode V11 as a factor
d$V12 <- as.factor(d$V12) #encode V12 as a factor
d$V13 <- as.factor(d$V13) #encode V13 as a factor
d$V14 <- factor(ifelse(d$V14 <1, 0, 1), levels = c(1,0)) #encode V14 as a fac
tor and specify classes

#name variables
names(d) <- c("age", "sex", "cp", "trestbps", "chol", "fbs", "restecg", "thal
ach", "exang", "oldpeak", "slope", "ca", "thal", "num")
summary(d) #print summary of d dataset
```

Objective A - Assess the performance of three different classification methods

Classification Method 1 - Logistic regression with ridge regression regularization

```
x <- model.matrix(num~.,d)[,-1] #specify the predictor variables V1-13
y <- d$num #specify the outcome variable V14 (num)

train.I <- sample(nrow(d),round(nrow(d)*2/3)) #split data into train and test
train.data <- d[train.I,] #specify training dataset
test.data <- d[-train.I,] #specify test dataset
train.x <- model.matrix(num~.,train.data)[,-1] #specify training x
test.x <- model.matrix(num~.,test.data)[,-1] #specify test x
train.y <- train.data$num #specify training y
```

```
test.y <- test.data$num #specify test y

cv.rr <- cv.glmnet(train.x, train.y, family="binomial", alpha=0, nfolds=10) #perform 10-fold cross-validation
cv.rr$lambda.min #print the lambda associated with the min MSE

pred.rr <- predict(cv.rr, newx=test.x, s=cv.rr$lambda.min, type="class") #create an object with the predicted classes using the the lambda associated with the min MSE
t.rr <- table(pred.rr, test.y)[2:1,2:1] #create confusion matrix of predicted class vs. true class
epi.tests(t.rr) #calculate sensitivity and specificity
#1 - optimal lambda = 0.03839479, sensitivity = 0.15, specificity = 0.19
#2 - optimal lambda = 0.0322885, sensitivity = 0.08, specificity = 0.26
#3 - optimal lambda = 0.04334681, sensitivity = 0.13, specificity = 0.21
#4 - optimal lambda = 0.04483221, sensitivity = 0.10, specificity = 0.29
#5 - optimal lambda = 0.03916323, sensitivity = 0.15, specificity = 0.17
```

Classification Method 2 - K-nearest neighbors

```
x <- model.matrix(num~.,data=d)[,-1] #specify the predictor variables V1-13
x <- scale(x) #standardize the predictor variables to make things comparable
y <- d$num #specify the outcome variable V14 (num)

train.I <- sample(nrow(d),round(nrow(d)*2/3)) #split data into train and test
train.data <- d[train.I,] #specify training dataset
test.data <- d[-train.I,] #specify test dataset
train.x <- model.matrix(num~.,train.data)[,-1] #specify training x
test.x <- model.matrix(num~.,test.data)[,-1] #specify test x
train.y <- train.data$num #specify training y
test.y <- test.data$num #specify test y

#KNN (k=1)
pred.knn <- knn(train.x, test.x, train.y, k=1) #create an object with the predicted classes
t <- table(pred.knn, test.y)[2:1,2:1] #create confusion matrix of predicted class vs. true class
epi.tests(t)
#1 - sensitivity = 0.56, specificity = 0.60
#2 - sensitivity = 0.60, specificity = 0.60
#3 - sensitivity = 0.60, specificity = 0.55
#4 - sensitivity = 0.56, specificity = 0.55
#5 - sensitivity = 0.60, specificity = 0.55

#KNN (k=5)
pred.knn <- knn(train.x, test.x, train.y, k=5) #create an object with the predicted classes
t <- table(pred.knn, test.y)[2:1,2:1] #create confusion matrix of predicted class vs. true class
epi.tests(t)
```

```
#1 - sensitivity = 0.51, specificity = 0.60
#2 - sensitivity = 0.77, specificity = 0.50
#3 - sensitivity = 0.56, specificity = 0.64
#4 - sensitivity = 0.54, specificity = 0.62
#5 - sensitivity = 0.70, specificity = 0.57

#KNN (k=10)
pred.knn <- knn(train.x, test.x, train.y, k=10) #create an object with the pr
edicted classes
t <- table(pred.knn, test.y)[2:1,2:1] #create confusion matrix of predicted c
lass vs. true class
epi.tests(t)
#1 - sensitivity = 0.65, specificity = 0.57
#2 - sensitivity = 0.81, specificity = 0.48
#3 - sensitivity = 0.60, specificity = 0.64
#4 - sensitivity = 0.63, specificity = 0.72
#5 - sensitivity = 0.78, specificity = 0.53

#KNN - Final - Choose k based on highest sensitivity and specificity
pred.knn <- knn(train.x, test.x, train.y, k=10) #create an object with the pr
edicted classes
t.knn <- table(pred.knn, test.y)[2:1,2:1] #create confusion matrix of predict
ed class vs. true class
epi.tests(t.knn) #calculate sensitivity and specificity
#1 - chosen k = 10, sensitivity = 0.58, specificity = 0.55
#2 - chosen k = 10, sensitivity = 0.81, specificity = 0.50
#3 - chosen k = 10, sensitivity = 0.65, specificity = 0.61
#4 - chosen k = 10, sensitivity = 0.61, specificity = 0.62
#5 - chosen k = 10, sensitivity = 0.72, specificity = 0.61
```

Classification Method 3 - Classification tree

```
x <- model.matrix(num~.,d)[,-1] #specify the predictor variables V1-13
y <- d$num #specify the outcome variable V14 (num)

train.I <- sample(nrow(d),round(nrow(d)*2/3)) #split data into train and test

train.data <- d[train.I,] #specify training dataset
test.data <- d[-train.I,] #specify test dataset
train.x <- model.matrix(num~.,train.data)[,-1] #specify training x
test.x <- model.matrix(num~.,test.data)[,-1] #specify test x
train.y <- train.data$num #specify training y
test.y <- test.data$num #specify test y

cfit <- tree(num~.,train.data) #fit the classification tree
cv.res <- cv.tree(cfit, FUN=prune.tree, method="misclass", K=5) #use cross-va
lidation to generate the pruned tree with the FUN=prune.tree argument; parame
ter k (not related to the k number of folds) = alpha
cv.res$size[which.min(cv.res$dev)] #print the size (number of terminal nodes)
that minimizes the misclassification error = 6
```

```
cfit.pruned <- prune.misclass(cfit, best=cv.res$size[which.min(cv.res$dev)])
#using the prune.misclass function, prune the tree with the optimal size

pred.c <- predict(cfit.pruned, newdata=test.data, type="class") #create an ob
ject with the predicted classes
t.c <- table(pred.c, test.y)[2:1,2:1] #create confusion matrix of predicted c
lass vs. true class
epi.tests(t.c) #calculate sensitivity and specificity
#1 - optimal size = 6, sensitivity = 0.79, specificity = 0.62
#2 - optimal size = 9, sensitivity = 0.90, specificity = 0.70
#3 - optimal size = 7, sensitivity = 0.89, specificity = 0.74
#4 - optimal size = 6, sensitivity = 0.84, specificity = 0.80
#5 - optimal size = 7, sensitivity = 0.76, specificity = 0.78
```

Objective B - Train a model that can be used for classification of future observations

Chosen Method - Classification tree

```
x <- model.matrix(num~.,d)[,-1] #specify the predictor variables V1-13
y <- d$num #specify the outcome variable V14 (num)

cfit <- tree(num~.,d) #fit the classification tree
plot(cfit) #plot the decision tree
text(cfit, pretty=0) #add text to the decision tree
title(main="Unpruned Classification Tree") #add title

cv.res <- cv.tree(cfit, FUN=prune.tree, method="misclass", K=5) #use cross-va
lidation to generate the pruned tree with the FUN=prune.tree argument; parame
ter k (not related to the k number of folds) = alpha
cv.res$size[which.min(cv.res$dev)] #print the size (number of terminal nodes)
that minimizes the misclassification error = 6
cfit.pruned <- prune.misclass(cfit, best=cv.res$size[which.min(cv.res$dev)])
#using the prune.misclass function, prune the tree with the optimal size
plot(cfit.pruned) #plot the pruned decision tree
text(cfit.pruned, pretty=0) #add text to the decision tree
title(main="Pruned Classification Tree")
```