2021

Heart Failure Analysis

MACHINE LEARNING FINAL PROJECT

O'NEILL, KYLE P.

Intro:

The dataset chosen is comprised of values that are commonly viewed as indicators for heart failure. In this project I will utilize machine learning techniques and algorithms to predict the possibility of a heart failure and gain insight into the data given the observations in the dataset utilizing the statistical coding software R. The algorithms used below are Principal Component Analysis, Decision Tree, Generalized Linear Model, Logistic Model, K nearest neighbors, and support vector machines. Both supervised and unsupervised learning techniques were implemented to gain deep insight into the dataset. Also, both regression and classification techniques were used.

Data Explanation:

This dataset consists of 299 observations with 13 variables of patients who had heart failure. The thirteen variables are as follows:

Age: The age of the patient (year)

Anemia: A decrease of red blood cells or hemoglobin (Boolean)

Creatinine Phosphokinase (CPK): level of the CPK enzyme in the blood (mcg/L)

Diabetes: if the patient has diabetes (Boolean) 0 indicates no 1 indicates yes

Ejection Fraction: percentage of blood leaving the heart at each contraction (percentage)

Platelets: Platelets in the blood (kiloplatelets/mL)

Sex: Woman or Male (Boolean) 0 indicated woman 1 indicates male

Serum creatinine: level of serum creatinine in the blood (mg/dL)

Serum sodium: level of serum sodium in the blood (mEq/L)

Smoking: If the patient smokes or not (Boolean) 0 indicates no 1 indicates yes

Time: follow up time between doctor visits (days)

Death Event: if the patient is deceased during the follow up period (Boolean) 0 indicates no 1 indicates yes

In this data set we are going to create models that will help predict the Death Event of a patient.

PCA (Principal Component Analysis):

Principal Component Analysis is a form of unsupervised learning. Unsupervised learning is one type of Machine Learning, the other being supervised learning. Before explaining what unsupervised learning is, we will tackle what supervised learning is first. For supervised

learning, you are given a dataset of (n) number of observations each with (p) number of predictors and your main goal is to make a prediction of one of the variables with the remaining variables you have. That prediction is called a response. You use the variables called predictors to create models that can tell give you a value of the response or a class for the response. For unsupervised learning, the goal is not to predict a number or an outcome, the goal is to find insight into the data. In unsupervised learning there is no response being predicted, but we are trying to find if there is some previously unknown relationships between variables or if there is a better way to view the data.

Principle Component Analysis is a great machine learning technique to apply to a dataset as the first thing you do to gain insight into the data. The goal of Principle Component Analysis is to use the predictors that we have and create linear combinations of those predictors called Principal Components that help explain the most amount of variance in the data as possible. Variance is the spread of the data, and if we can explain more of the spread, we can explain more datapoints. So, in this dataset we have 13 variables, so running PCA will give us thirteen Principal Components. The first principal component will cover the most amount of variance, the second will cover the second most, and so on and so forth. This process will sometimes allow for a reduction in the dimensionality of the dataset. It is possible that the first few principal components will explain most of the data, which leads to a reduction in dimensionality which helps simplify the data.

For this dataset to implement PCA we need to scale the data first. This makes it so that any variables with high values do not have unrealistic representation in the dataset. Scaling the data simply entails taking the mean of each column, each predictor as a vector, and subtracting that mean from each observation value in that predictor. Then you divide by the standard deviation of the column and the data is set for PCA. For this dataset specifically, implementing PCA does not show a great improvement in dimensionality. The table below shows the amount of data each principal component represents.

PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12	PC13
15.62	12.78	10.09	8.91	7.93	7.70	6.99	6.50	5.91	5.45	5.23	3.95	2.90

As you can see the first principal component only accounts for 15.62% of the data, and each principal component is only slightly higher than the one after it. This would indicate that there is no reason to drastically reduce the dimension of the dataset.

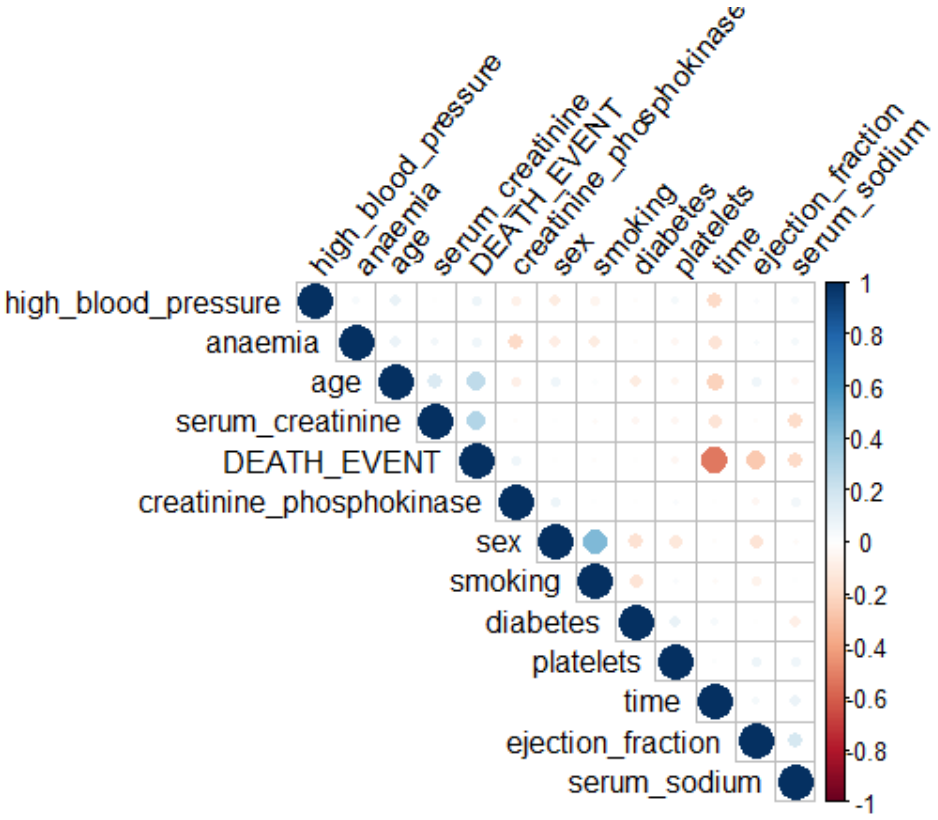
Simple Multiple Regression Model:

It is the goal of this project to predict the Death Event of the patient. It can help to understand which variables will be useful in doing so. Running a simple linear model on the dataset with Death Event as the response will allow us to see which variables will be considered in the future when trying to predict the outcome. The information we get when running a simple multiple regression model shows us that Time, Ejection Fraction, and Serum Creatinine are the

only variables that we can consider when creating prediction models for Death Event. We can assume this because these are the only variables that have a p-value below 0.05. This p-value allows us to be at least 95% confident in the data that this variable represents. These are the top three values that explain Death Event so we will be using them in future models. To further visualize the relationship between Death Event and the other variables, I created a covariance heat map that illustrates the relationship between all the variables. In this heat map we can see that

Co-variance is the relationship that two variables have with each other. So, in the case of Death Event and the rest of the variables, the variables that correlate the most are time, ejection fraction, serum creatinine. As we covered before, these are the only variables that are worth trusting in predicting Death Event. Covariance is the measure of variation between two variables, so how much the spread of one affects the spread of another. When co-variance is closer to positive 1, as one variable increase the other increases. When covariance is closer to negative 1, as one variable increase the other decreases. So, the relationship for variables with a negative covariance is inverse.

As you can see in this graph the dots on the diagonal are all blue, that is because in a covariance matrix the diagonal is just the variance of a point with itself, which is equal to one. As stated above, a variance of one indicates that as one variable increase the other increase, and the variance of a variable and itself will always be one. The top three predictors that have the highest correlation with Death Event are time, ejection fraction, and serum creatinine.



Generalized Linear Model (GLM):

Now that we have a basic understanding of our data, we can implement a training model to predict the outcome of Death Event. For Death Event, the top three variables that had a low enough p-value and were good fits for prediction were Creatinine Serum, Ejection Fraction, and Time. These were the only three variables I accounted for in this model. The GLM model is a logistic regression model, so the response of the model will be a probability that an observation is in a specific class. Logistic regression calculates the maximum likelihood that an observation is associated with a specific class. This is a supervised learning technique, so I created a testing set of 100 observations out of the 299 total observations and set them aside to test the GLM model I create. The remaining 199 observations were used to train my model. As Death Event is a Boolean response, the outcome of the model will give a percentage that an observation is the Death Event class associated with 1, which is a death occurring. So for our data is there is an over 50% chance of a Death Event occurring, I assigned that observation into the class 1, which represents a Death Event occurring.

The model created is then tested with the 100 variables that I set aside in the beginning, as we already know what class they fall into we can see how accurate my GLM model was at predicting that class. Out of the 100 variables used as a test run, 13 were misclassified and 87 were correctly classified. This would give us a test error rate of 13%. Therefore, we can assume that if we are given the Time, Serum Creatinine levels, and Ejection Fraction percent of an observation, we can make a correct prediction 87% of the time as if a Death Event occurred using a GLM.

Linear Discriminant Analysis:

The next type of supervised model that we can create is called a Linear Discriminant Analysis model, otherwise know as LDA. An LDA model is similar to a logistic regression model, with a few key differences. LDA can be beneficial to use if the observations and predictors are not normally distributed or if there is a small number of total observations. We can perform a LDA model to predict the Death Event with the variables Ejection Fraction, Time, and Creatinine Serum and compare it with the GLM model we just made.

One key output of Linear Discriminant Analysis model is the probability associated with each variable and each category. As we are trying to predict the Death Event, our two categories are 0 and 1. 0 represents a death not occurring and 1 represents a death occurring. LDA gives us the probability that a single point is in each of these categories. For example, one random point might have a 25% probability of being classified as 0 and a 75% chance of being classified as a 1. In the model I created, this random point would be classified as a 1 as I made the threshold for classification 50%. So if a point has a more than 50% probability of being in a specific category, either a 0 or 1, it will be classified into that specific category.

Similar to the GLM above I took a test sample of 100 variables and held them off to the side to use later and see how accurate the model I created was. With the remaining 199 variables,

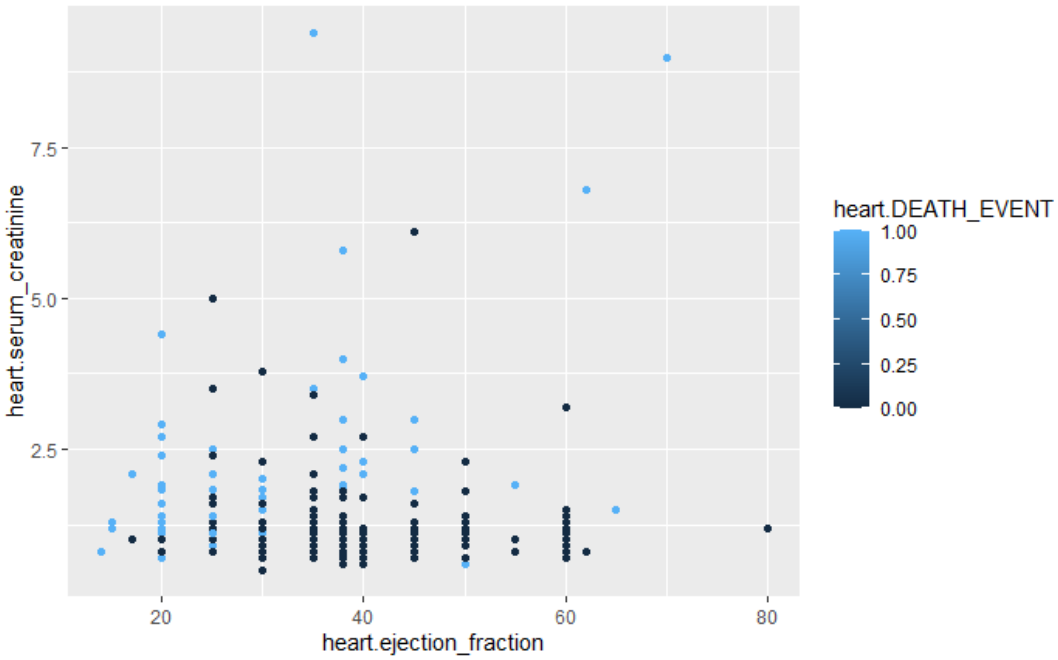
I created a model that will attempt to predict if given the time, creatine serum level, and ejection fraction percent if a Death Event occurred. After creating the model, we can see the accuracy by comparing the results of the 100 test variables with their actual classification, as we already know the classification of each of these points. When comparing the models output with the known Death Event Classification, we can see that of the 100 test variables 87 of the variables were correctly classified, and 13 of the variables were incorrectly classified. This gives the test error rate of 13%. This is the same test error rate of the GLM model that I made previously.

K Nearest Neighbors (KNN):

This type of machine learning technique is different from the previous two as it is a classification technique and not a regression technique. For regression, the output is a quantity. So, for logistic regression the output is a probability that can help us assign an observation to specific groups. For classification, the output is a category or a group. By implementing a KNN model, we can use already known points and their classifications to predict the classification of new observations.

K nearest neighbors is a simple classification problem to understand. Say we plot all our points on a graph, and we set k=1. K is the number of already known observations that will help us predict a new point. If k=1, then we will take the classification of the single closest point to the new observation and assign the new observation to the same class as that single closest point. If we set k=5, then we would take the five closets' points to the new observation and take the majority class and assign that to the new observation.

This plot can help visualize what KNN does. This is a plot of every observation in our data, with only the Serum Creatinine levels and the Ejection Fraction percent shown with their associated Death Event output. Dark blue indicates a Death Event in the category 0, so in other words no death occurring. Light blue represents a variable with the Death Event category of 1, so a death did occur. If we were to take a brand-new observation and plot it on the graph with k=1, we would take the next closest point and assign the new observation to the same category as that closest point we just found. The model I created is a little more complex as I used Serum Creatinine, Ejection fraction, and time to predict the category of a new observation, but the idea stays the same.



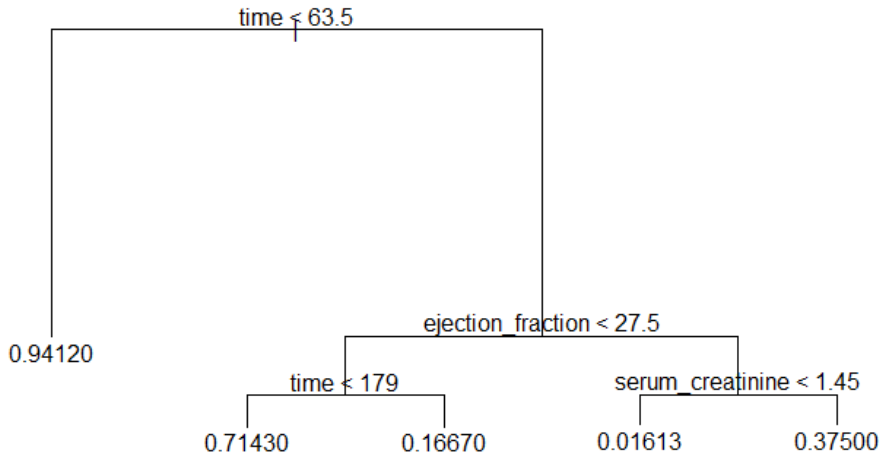
For this model again I kept 100 variables off to the side to use I order to test the accuracy of the KNN model I created against their actual classification. Then I ran 3 models, one with k=1, another with k=2, again with k=3, then finally with k=5. For k=1 I got a classification error rate of 14%. When k=2 I got a classification error rate of 10%. When k=3 I got a classification error rate of 11%. When k=5 I got a classification error rate of 9%. These error rates are similar to the regression models used above.

Decision Tree:

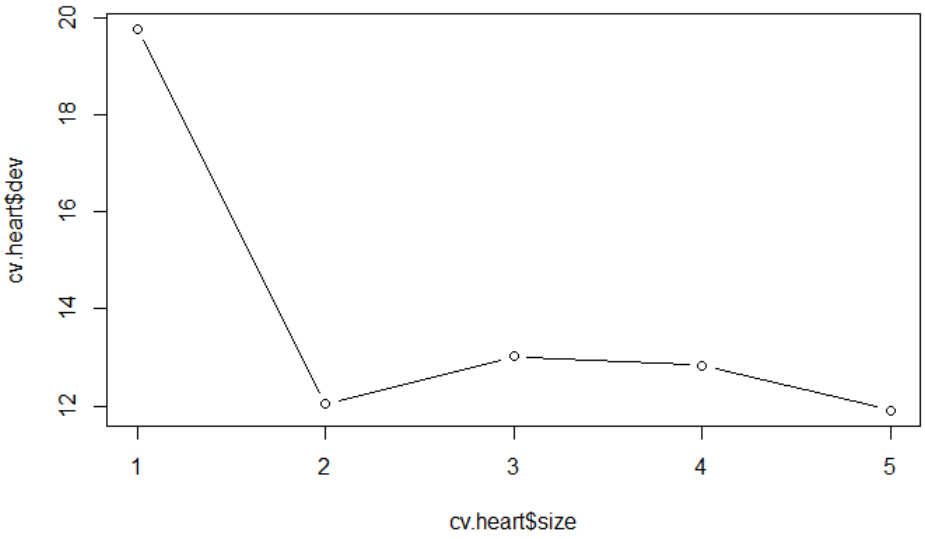
A decision tree can be one of the simplest machine learning techniques to interpret. Before we get to how I implemented my decision tree, lets quickly go over how they work. A new observation will be assigned into a specific sub region of a decision tree, this will be based on the qualities and values that the new observation holds. A decision tree will take the average of the sub region and assign that value to the new observation, which is the predicted value of the new observation. These specific regions are created in a greedy fashion. This means that each split of a decision tree will be done to optimize the RSS, Residual of Squared Sums. The RSS of an observation is the observations value, which is previously known as this is supervised learning, subtracted from the mean of the specific subregion and then that difference is squared. The goal is to keep that value as low as possible. This method is called recursive binary splitting. This is choosing the split that is has the best reduction in RSS then splitting that region and repeating this process several times. At each stage we choose the best split, which is why it is called a greedy algorithm.

A decision tree can be used as a classification or a regression model. In this case, I used it as a regression model. In a regression tree the output is a number. This means that the value of a node will be the average value of the response in that specific group. So, for the tree below, if we take the group that is Time and below 63.5, the average value for death event is 0.94120. Another example would be if time is greater than 65.5, and ejection fraction is greater than 27.5, and serum creatinine is greater than 1.45, the average value of Death Event of this specific group is .375. As the response Death Event is a binary variable, the response will be a value between 0 and 1. Each node of a regression tree is the average of the response associated with that specific subregion.

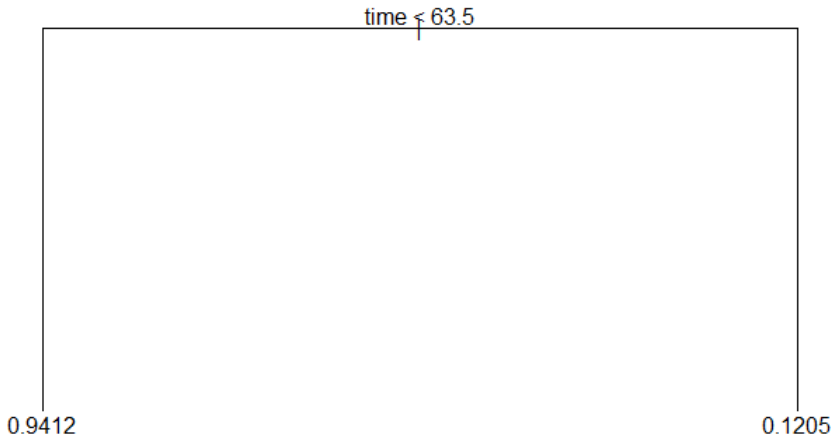
The tree below was trained on the same 199 variables as the other models created and tested on the same 100 observations. The test error rate for this tree was about 14.06%. This means that this model is about 86% accurate when predicting the Death Event given these parameters. Therefore, a new observation with specific qualities assigned to a specific node will be predicted correctly about 86% of the time.



As we have made a significantly large enough tree, we can begin to prune it back and remove branches that do not have a significant impact in the RSS and keep the most impactful cuts. Pruning a decision tree is done to reduce the dimensionality of the tree and reduce overfitting. A tree with too many splits can be overfitted, which means it has a very high variance. Pruning the tree to splits that are the most impactful will help reduce the variance of the tree. We can choose the number of nodes we want the tree to be pruned to by cross validating. The plot below shows the variance associated with a number of nodes. We can see that at two nodes the variance looks to be the lowest, so we will prune the tree back to two nodes.



The cross-validation plot shows us that our RSS reaches a low at two branches. Therefore, we can prune the tree back to two branches and still maintain a significant amount of data being represented. Again, we can test the error rate. We see that the test error rate has gone up to about 17%. This is an increase but a very minor one as we consider that the number of nodes in the tree has been reduced a great amount. The only split we maintain is time being less than 63.5 or time being grater than 63.5. The average value of the Death Event for Time being less than 63.5 is .9412, which is very close to a value of 1, which indicates a death event. The average value when time is above 63.5 is .1205, which is very close to the value of 0, which indicates no death occurring. This can be seen in the pruned tree below.



Support Vector Machine (SVM):

A Support Vector Machine can be a very powerful classification technique. SVM's use hyperplanes to classify datapoints into two categories. Hyperplanes are linear higher dimension planes that separate the data into two halves. In two dimensions (two predictors) this would look like a line through a plot of data where all the observations above the line are in one class and all the data below the line are in another class. If the data can be separated with a hyperplane, then we use the maximal margin technique to find the best hyperplane possible. This is done for a given hyperplane by maximizing the size of the margin, where a margin is the distance of the points closest to the hyperplane. This is done through an optimization technique. The observations that control the location of the maximal margin hyperplane are called the support vectors.

It can be the case however that a hyperplane does not perfectly separate the dataset into two distinct classes, or that a hyperplane that does separate the data into two distinct classes will incorrectly classify new observations more often than one which allows for some initial misclassifications in order to get a better overall fit. A support vector classifier helps for both these cases as it allows for some initial misclassifications in order to better predict future points. SVC's with soft margins can be useful especially when there are outliers in the data. Soft margins allow for some misclassifications whereas the hard margins of a maximal margin hyperplane allow for no misclassifications.

There are multiple types of classification line a SVM can use, these are called kernels. A linear kernel is a straight line that separates the data. A polynomial kernel is a curved line that separates the data. A radial line is a circle that separates the data. An SVM with a radial kernel seems like it would fit this data the best. That is, a Support Vector Machine that does not use lines to classify boundaries but uses circles. It gives the lowest error rate of 11%.

Conclusion:

There are many machine learning techniques that can be applied to this dataset to help us both understand how different variables in the data correlate with each other and how certain variables can be used to predict the response of other variables. In this document the main focus was predicting the Death Event, which was whether a patient died in the time between doctor appointments. We found that the top three variables that had a significant impact on the Death Event were Time, Creatinine Serum levels, and Ejection Fraction percent. Both classification and regression techniques can be applied in order to predict the occurrence of a Death Event. Both supervised and unsupervised learning techniques were implemented to discover more about the data.

What we learned from this data is that we can accurately predict if a new observation is deceased in the time period between follow up appointments with the variables Time, Ejection Fraction, and Serum Creatinine alone with about 90% accuracy depending on the machine learning technique being used. Using the model KNN and a value of k=5, we were able to produce the most accurate prediction technique with an accuracy of 91%. A pruned decision tree was the least accurate with an accuracy of 83%.

One take away from analyzing this data is that the results are very specific to the dataset. The variable time is not a predictor that can be used in a universal sense, as it is does not actually give insight into the health of a patient. Another way to analyze this data would have been to only use predictors that can transcend this dataset so that the results could be applied to anyone with knowledge of the intricacies of their heart related statistics.

R Code:

```
# this is to read in the cvs file
heart=read.csv("heart_failure_clinical_records_dataset.csv")
summary(heart)
names(heart)

# this is to create a PCA
pca.out=prcomp(heart, scale=TRUE)
plot(pca.out$x[,1:2])
summary(pca.out)

# this plot shows the a coorelation heat plot of all the variables in the data
res=cor(heart)
library(corrplot)
corrplot(res, type = "upper", order = "hclust", tl.col = "black", tl.srt = 50)

# this is a multiple regression model with all the variables with the responce Death Event
names(heart)
lm=lm(heart$DEATH_EVENT~., data=heart)
summary(lm)

#this is a multiple regression model with the top three predictors for death event
newlm=lm(heart$DEATH_EVENT~heart$ejection_fraction+heart$serum_creatinine+heart$time
)
summary(newlm)



# this is a decision tree
set.seed(25)
library(ISLR)
```

```
library(tree)
#newheart=heart[,-12], this gets rid of the time element
newheart=heart
summary(newheart)
train=sample(1:nrow(newheart),100)
heart.test=newheart[-train,]
tree.heart=tree(newheart$DEATH_EVENT~.,newheart, subset=train)
tree.pred=predict(tree.heart,heart.test)
summary(tree.heart)
plot(tree.heart)
text(tree.heart, pretty=1)

yhat=predict(tree.heart, newdata=heart.test)
mean(I(yhat-heart.test$DEATH_EVENT)^2)

# this is the cross validation plot of the decision tree
set.seed(2)
cv.heart=cv.tree(tree.heart)
plot(cv.heart$size,cv.heart$dev,type="b")

# this is to prune the tree to get a simpler result that is more efficient
prune.heart=prune.tree(tree.heart,best=2)
plot(prune.heart)
text(prune.heart, pretty = 0)

yhat=predict(prune.heart,newdata=heart.test)
mean(I(yhat-heart.test$DEATH_EVENT)^2)



# this is a linear support vector machine
```

```
library(e1071)
set.seed(4)

tune1=tune(svm,DEATH_EVENT~ejection_fraction+serum_creatinine+time, data=heart,
kernel="linear", ranges=list(cost=c(.001,0.01,0.1,1)))
summary(tune1)

tune2=tune(svm,DEATH_EVENT~ejection_fraction+serum_creatinine+time, data=heart,
kernel="polynomial", ranges=list(cost=c(.001,0.01,0.1,1), degree=c(2,3,4,5)))
summary(tune1)

tune3=tune(svm,DEATH_EVENT~ejection_fraction+serum_creatinine+time, data=heart,
kernel="radial", ranges=list(cost=c(.001,0.01,0.1,1), gamma=c(.001, 0.01,0.1,1)))
summary(tune3)



# this is a glm model that is trained and cross validated
set.seed(4)
summary(heart)
v= sort(sample(1:nrow(heart),100))
newdata.test=heart[v,]
newdata.train=heart[-v,]

glm.fits01=glm(heart$DEATH_EVENT~heart$ejection_fraction+heart$serum_creatinine+heart
$time,data=newdata.train,family=binomial)
glm.probs=predict(glm.fits01,type="response")
glm.pred=rep(0,199)
glm.pred[glm.probs[-v]>.5]=1
table(glm.pred,newdata.train$DEATH_EVENT)
mean(glm.pred==newdata.train$DEATH_EVENT)
```

```
glm.probs=predict(glm.fits01, heart, type="response")
glm.pred=rep(0,length(heart$DEATH_EVENT))
glm.pred[glm.probs >.5]=1
table(newdata.test$DEATH_EVENT,glm.pred[v])
test.error.rate=mean(glm.pred[v]!=newdata.test$DEATH_EVENT)
test.error.rate
length(glm.pred[v])



# this is an lda that is also cross validated
library(MASS)
lda.fits=lda(heart$DEATH_EVENT~heart$ejection_fraction+heart$serum_creatinine+heart$tim
e, data=newdata.train)
lda.fits
lda.pred = predict(lda.fits, heart)
lda.class=lda.pred$class



sum(lda.pred$posterior[,1]>=.5)
sum(lda.pred$posterior[,1]<.5)
test.error.ratelda1=mean(lda.class[v]!=newdata.test$DEATH_EVENT)
test.error.ratelda1

table(lda.class[v],newdata.test$DEATH_EVENT)



# this is knn that is trained but not cross validated
library(class)
train.X=cbind(heart$ejection_fraction,heart$serum_creatinine,heart$time)[-v,]
```

```
test.X=cbind(heart$ejection_fraction,heart$serum_creatinine,heart$time)[v,]
train.de=heart$DEATH_EVENT[-v]

knn.pred01=knn(train.X, test.X, train.de, k=1)
knn_ter01=mean(knn.pred01 != heart$DEATH_EVENT[v])
knn_ter01

knn.pred02=knn(train.X, test.X, train.de, k=2)
knn_ter02=mean(knn.pred02 != heart$DEATH_EVENT[v])
knn_ter02

knn.pred03=knn(train.X, test.X, train.de, k=3)
knn_ter03=mean(knn.pred03 != heart$DEATH_EVENT[v])
knn_ter03

knn.pred04=knn(train.X, test.X, train.de, k=5)
knn_ter04=mean(knn.pred04 != heart$DEATH_EVENT[v])
knn_ter04

# this is to help with the visualization of KNN
matrix1=data.frame(heart$ejection_fraction,heart$serum_creatinine,heart$DEATH_EVENT)

#install.packages("ggplot2")
library(ggplot2)
#fix(matrix1)
ggplot(matrix1, aes(x=heart.ejection_fraction,y=heart.serum_creatinine,
col=heart.DEATH_EVENT))+geom_point()
```

Link to Data: https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records