classification: knn, logreg

Lisa Skalon

6/9/2020

```r
library('ggplot2')
library('ggpubr')
library('dplyr')
library('tidyr')
library(class)
library(psych)
library(caret)
library(ROCR)
library(reshape2)
library(boot)
```

We are going to analyze open dataset with heart disease data, available at UCI (https://archive.ics.uci.edu/ml/datasets/Heart+Disease). The description of variables:

1. **age**: age in years
2. **sex**: sex (1 = male; 0 = female)
3. **cp**: chest pain type
    - Value 1: typical angina
    - Value 2: atypical angina
    - Value 3: non-anginal pain
    - Value 4: asymptomatic
4. **trestbps** : resting blood pressure (in mm Hg on admission to the hospital)
5. **chol**: serum cholestoral in mg/dl
6. **fbs**: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. **restecg**: resting electrocardiographic results
    - Value 0: normal
    - Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    - Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. **thalach**: maximum heart rate achieved
9. **exang**: exercise induced angina (1 = yes; 0 = no)
10. **oldpeak**: ST depression induced by exercise relative to rest
11. **slope**: the slope of the peak exercise ST segment
    - Value 1: upsloping
    - Value 2: flat
    - Value 3: downsloping
12. **ca**: number of major vessels (0-3) colored by flourosopy
13. **thal**: 3 = normal; 6 = fixed defect; 7 = reversable defect

14. **num**: 0 = no heart disease, > 0 = heart disease

The variable wich we are going to predict is **num** - the presence or the absence of heart disease.

We will use KNN algorithm, which classifies new cases based on a similarity measure (Euklidian distance, ..). The algorithm assumes that similar things exist in close proximity. The number of neighbors(K) is a tuned hyperparameter - we should manually choose the best K.

```
# read df
df <- read.csv("https://archive.ics.uci.edu/ml/machine-learning-databases/heart-disease/processed.clevel
names(df) <- c( "age", "sex", "cp", "trestbps", "chol","fbs", "restecg",
                   "thalach","exang", "oldpeak","slope", "ca", "thal", "num")

# deal with na
str(df)
```

```
## 'data.frame':    303 obs. of  14 variables:
##  $ age     : num  63 67 67 37 41 56 62 57 63 53 ...
##  $ sex     : num  1 1 1 1 0 1 0 0 1 1 ...
##  $ cp      : num  1 4 4 3 2 2 4 4 4 4 ...
##  $ trestbps: num  145 160 120 130 130 120 140 120 130 140 ...
##  $ chol    : num  233 286 229 250 204 236 268 354 254 203 ...
##  $ fbs     : num  1 0 0 0 0 0 0 0 0 1 ...
##  $ restecg : num  2 2 2 0 2 0 2 0 2 2 ...
##  $ thalach : num  150 108 129 187 172 178 160 163 147 155 ...
##  $ exang   : num  0 1 1 0 0 0 0 1 0 1 ...
##  $ oldpeak : num  2.3 1.5 2.6 3.5 1.4 0.8 3.6 0.6 1.4 3.1 ...
##  $ slope   : num  3 2 2 3 1 1 3 1 2 3 ...
##  $ ca      : num  0 3 2 0 0 0 2 0 1 0 ...
##  $ thal    : num  6 3 7 3 3 3 3 3 7 7 ...
##  $ num     : int  0 2 1 0 0 0 3 0 2 1 ...
```

```
sum(is.na(df))
```

```
## [1] 6
```

```
df <- drop_na(df)

# change num to binary as in the description
df$num[df$num > 0] <- 1

df_glm <- df

# change data types
df <- transform(df, trestbps = as.factor(sex), cp = as.factor(cp), fbs = as.factor(fbs),
                exang = as.factor(exang), restecg = as.factor(restecg),
                slope = as.factor(slope), ca = as.factor(ca), thal=as.factor(thal),
                sex = as.factor(sex), num=as.factor(num))


summary(df)
```

```
##       age         sex     cp      trestbps      chol       fbs     restecg
##  Min.   :29.00   0: 96   1: 23   0: 96    Min.   :126.0   0:254   0:147
##  1st Qu.:48.00   1:201   2: 49   1:201    1st Qu.:211.0   1: 43   1:  4
##  Median :56.00           3: 83            Median :243.0           2:146
##  Mean   :54.54           4:142            Mean   :247.4
##  3rd Qu.:61.00                            3rd Qu.:276.0
##  Max.   :77.00                            Max.   :564.0
##     thalach       exang      oldpeak      slope    ca       thal    num
##  Min.   : 71.0   0:200   Min.   :0.000   1:139   0:174   3:164   0:160
##  1st Qu.:133.0   1: 97   1st Qu.:0.000   2:137   1: 65   6: 18   1:137
##  Median :153.0           Median :0.800   3: 21   2: 38   7:115
##  Mean   :149.6           Mean   :1.056           3: 20
##  3rd Qu.:166.0           3rd Qu.:1.600
##  Max.   :202.0           Max.   :6.200
```

```
# check the number of diseased/healthy obs
table(df$num)
```

```
##
##   0   1
## 160 137
```

We will normalize numeric features, because they are in different scales

```
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }

num_columns <- sapply(df, is.numeric)
df[ , num_columns] <- lapply(df[ , num_columns], normalize)
```

KNN requires all variables besides the predictor variable to be numeric. We need to dummy code any categorical variables.

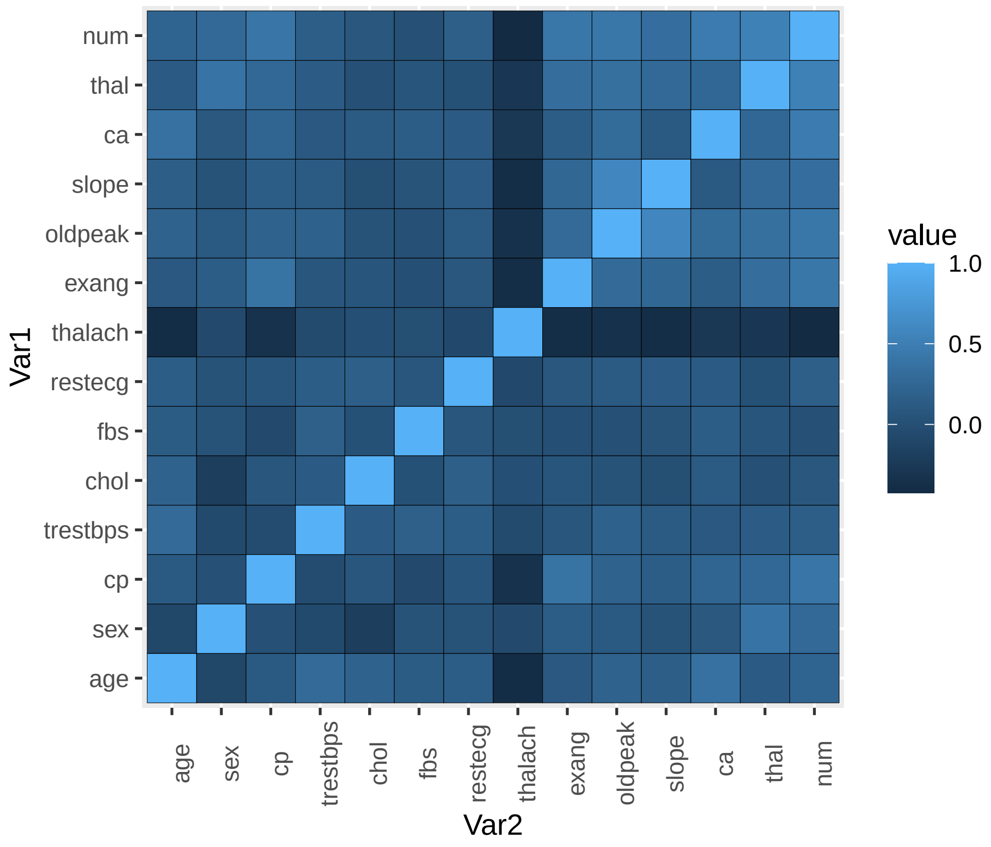
```
# make dummies
factor_columns <- sapply(df[-14], is.factor)
df[ , factor_columns] <- lapply(df[ , factor_columns] , dummy.code)
```

Correlation matrix

```
cor_mtx <- (cor(df_glm))
head(cor_mtx)
```

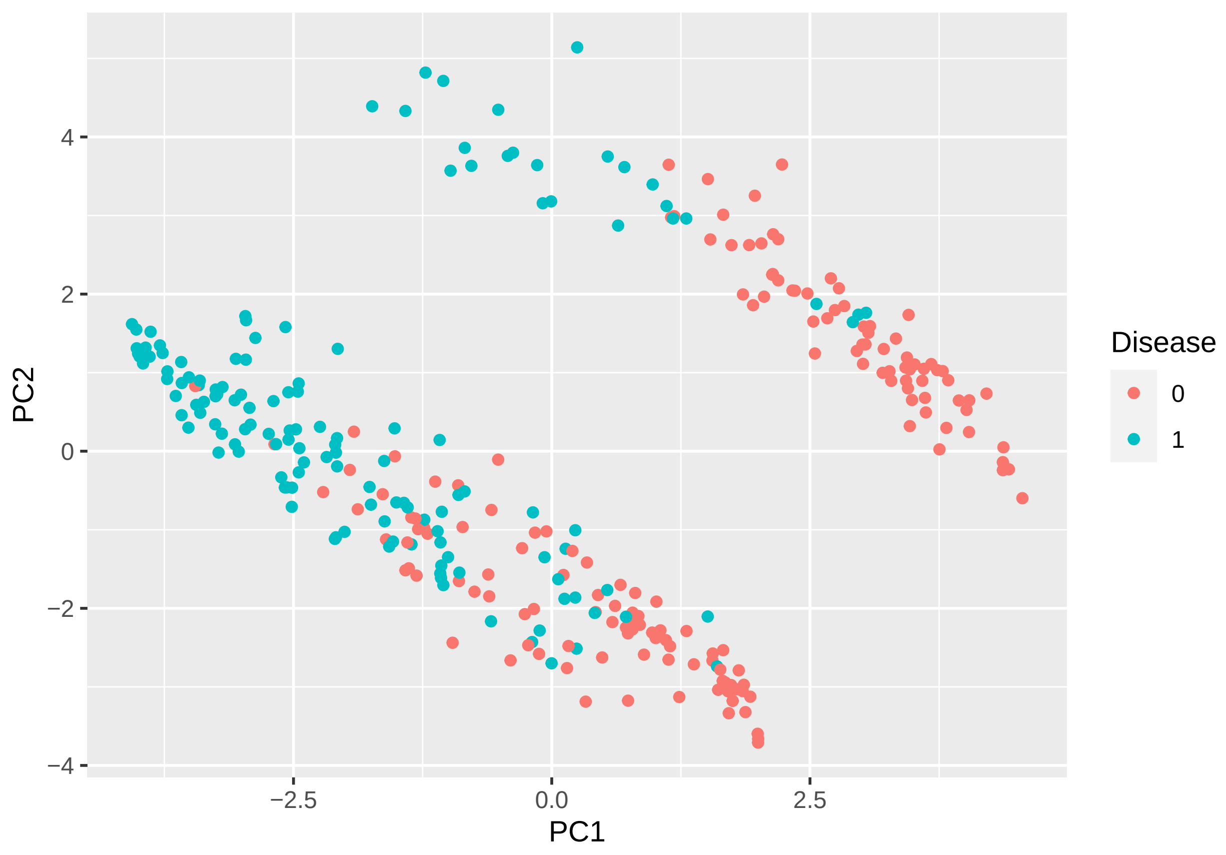
```
##                 age          sex           cp    trestbps        chol
## age       1.00000000 -0.092399479  0.110470866  0.29047626  0.20264355
## sex      -0.09239948  1.000000000  0.008908026 -0.06634020 -0.19808906
## cp        0.11047087  0.008908026  1.000000000 -0.03697975  0.07208831
## trestbps  0.29047626 -0.066340200 -0.036979748  1.00000000  0.13153571
## chol      0.20264355 -0.198089063  0.072088310  0.13153571  1.00000000
## fbs       0.13206199  0.038850300 -0.057663110  0.18085954  0.01270828
##                 fbs    restecg       thalach        exang     oldpeak
## age       0.13206199 0.14991651 -3.945629e-01  0.0964888046 0.197122616
## sex       0.03885030 0.03389683 -6.049601e-02  0.1435812504 0.106567243
## cp       -0.05766311 0.06390469 -3.393076e-01  0.3775247891 0.203243824
## trestbps  0.18085954 0.14924228 -4.910766e-02  0.0666910687 0.191243136
## chol      0.01270828 0.16504603 -7.456799e-05  0.0593389323 0.038595794
## fbs       1.00000000 0.06883111 -7.842359e-03 -0.0008930821 0.008310667
##               slope         ca       thal        num
## age       0.15940474 0.36221034 0.12658600 0.22707515
## sex       0.03334496 0.09192480 0.38365175 0.27846670
## cp        0.15107859 0.23564412 0.26849955 0.40894469
## trestbps  0.12117205 0.09795376 0.13818322 0.15349003
## chol     -0.00921524 0.11594459 0.01085909 0.08028475
## fbs       0.04781901 0.15208589 0.06220901 0.00316683
```

```
# heatmap
ggplot(data = melt(cor_mtx), aes(Var2, Var1, fill = value))+
 geom_tile(color = "black")+
 theme(axis.text.x = element_text(angle = 90))+
  coord_fixed()
```



PCA plot to check the clusters

```
prc <- prcomp(x = df[-14], center=TRUE, scale=T)
prc_adj <- data.frame(prc$x, Disease = df$num)
ggplot(data = prc_adj, aes(x = PC1, y = PC2, color = Disease)) +
    geom_point()
```



We are going to build knn model. We should pick an optimal value for k parameter. We could try some values, and then just choose the one which performs the best on our training data, in terms of the number of errors the algorithm would make if we apply it to the samples we have been given for training.

```
# divide into train and test set
set.seed(42)
sample <- sample.int(n = nrow(df), size = floor(.75*nrow(df)), replace = F)
train <- df[sample, ]
test  <- df[-sample, ]
```

Firstly, we just try a random K and calculate the accuracy of the prediction. Accuracy is the percentage of correctly classifies instances out of all instances.

```
pred_knn <- knn(train[1:13],
                test[1:13],

                cl = train$num,
                k = 7)

tab <- table(pred_knn, real = test$num)
print(tab)
```

```
##          real
## pred_knn  0  1
##         0 35  6
##         1 13 21
```

```r
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x))))}

# the prediction accuracy
accuracy(tab)
```

```
## [1] 0.7466667
```

Then we will use caret package. In this package, the function picks the optimal number of neighbors (k).

If we want to know what the best value for a tunable parameter is, we need to see how different values of the parameter perform on samples, which are not in the training data - we use cross-validation for that purpose.

CV use all the data for testing the predictive accuracy by splitting the data into a number of folds. If we have N folds, then the first step of the algorithm is to train the algorithm using (N-1) of the folds, and test the algorithm's accuracy on the single left-out fold. This is then repeated N times until each fold has been used as in the test set.

We often are underestimating the true error rate since our model has been forced to fit the test set in the best possible manner - CV solves this problem by estimating the test error rate by holding out a subset of the training set from the fitting process.

The best K is the one that corresponds to the lowest test error rate, so let's suppose we carry out repeated measurements of the test error for different values of K.

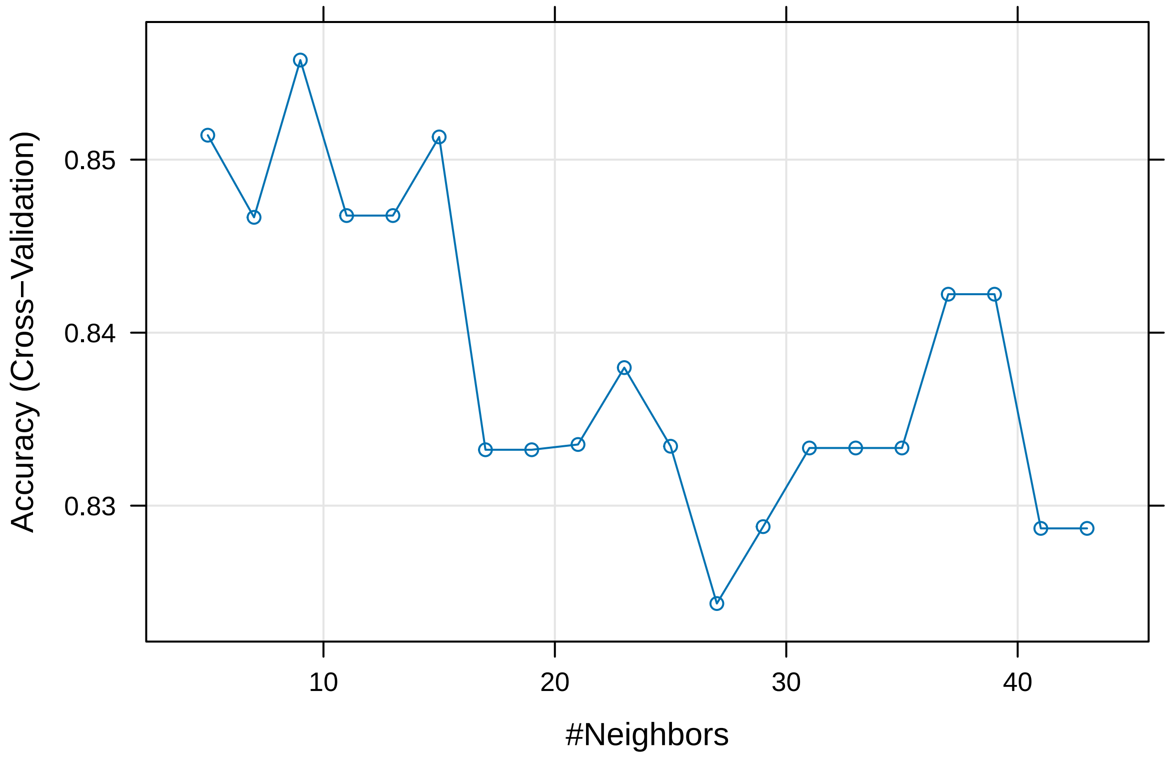
We will use the createFolds function from the caret package to make 5 folds

```r
trControl <- trainControl(method  = "cv",
                          number  = 5)

# caret model with 5 folds cv and 20 k checked
knn_caret <- train(train[1:13], train$num, method = "knn", preProcess = c("center","scale"), trControl
knn_caret
```

```
## k-Nearest Neighbors
##
## 222 samples
##  13 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (4), scaled (4), ignore (9)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 178, 178, 177, 177, 178
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8514141  0.7028718
##    7  0.8466667  0.6935607
##    9  0.8557576  0.7116028
##   11  0.8467677  0.6936625
##   13  0.8467677  0.6936625
##   15  0.8513131  0.7026655
##   17  0.8332323  0.6665435
##   19  0.8332323  0.6664378
##   21  0.8335354  0.6669345
##   23  0.8379798  0.6758893
##   25  0.8334343  0.6666928
##   27  0.8243434  0.6486340
##   29  0.8287879  0.6575889
##   31  0.8333333  0.6665741
##   33  0.8333333  0.6665741
##   35  0.8333333  0.6665741
##   37  0.8422222  0.6844837
##   39  0.8422222  0.6844837
##   41  0.8286869  0.6573473
##   43  0.8286869  0.6572418
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
plot(knn_caret)
```



Now we will try caret model on test data. Kappa parameter is like classification accuracy, except that it is normalized at the baseline of random chance.

```
knnPredict <- predict(knn_caret, newdata = test[-14] )
#Get the confusion matrix to see accuracy value and other parameter values
confusionMatrix(knnPredict, test$num )
```

```
## Confusion Matrix and Statistics
##
```

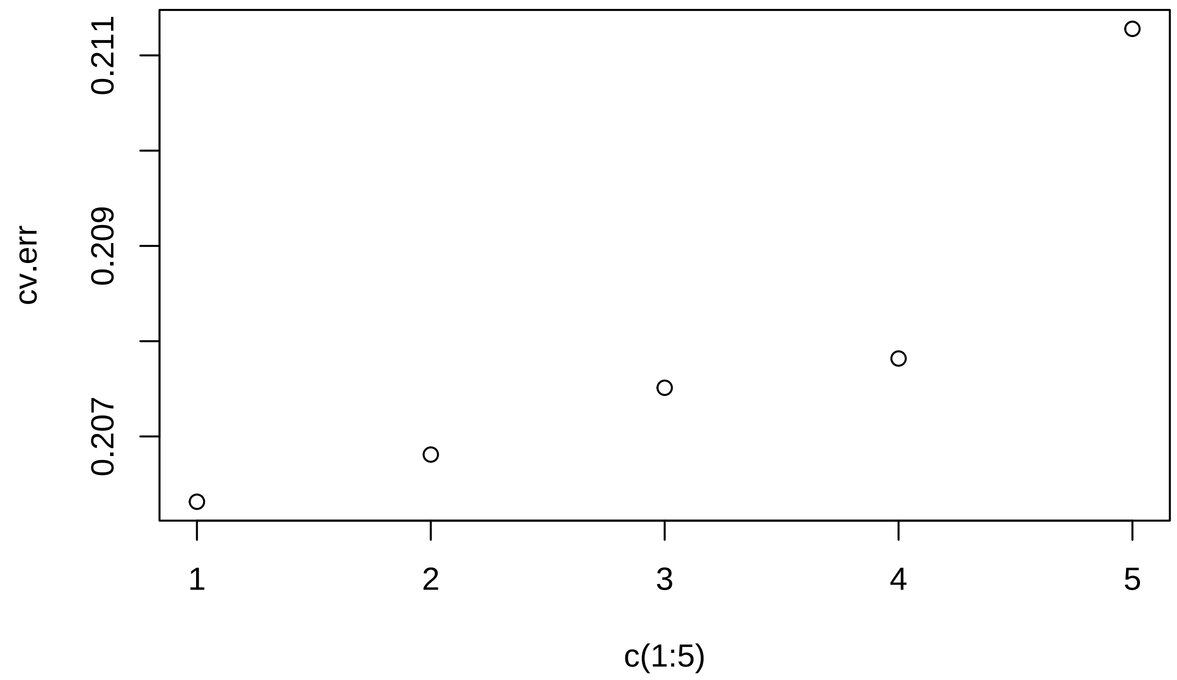
```
##           Reference
## Prediction  0  1
##          0 35  4
##          1 13 23
##
##                Accuracy : 0.7733
##                  95% CI : (0.6621, 0.8621)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.009393
##
##                   Kappa : 0.5415
##
##  Mcnemar's Test P-Value : 0.052345
##
##             Sensitivity : 0.7292
##             Specificity : 0.8519
##          Pos Pred Value : 0.8974
##          Neg Pred Value : 0.6389
##              Prevalence : 0.6400
##          Detection Rate : 0.4667
##    Detection Prevalence : 0.5200
##       Balanced Accuracy : 0.7905
##
##        'Positive' Class : 0
##
```

We will use polynomial regression algorithm as well. We can try to model nonlinear relationships. The degree of the polynomial is tuned hyperparameter as well. So we will use CV to try different degrees

```
cv.err <- rep(0, 5)
for (i in 1:5) {
    gl <- glm(num ~ poly(thalach, i), data = df_glm)
    cv.err[i] <- cv.glm(df_glm, gl)$delta[1]
}
cv.err
```

```
## [1] 0.2063146 0.2068096 0.2075110 0.2078180 0.2112786
```

```
plot(x=c(1:5), y=cv.err )
```



The best degree value is 1 (no degree. Let's design the model

```
fit3 <- glm(num ~ poly(thalach, 1), df_glm, family = 'binomial')
summary(fit3)
```

```
##
## Call:
## glm(formula = num ~ poly(thalach, 1), family = "binomial", data = df_glm)
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.1569     0.1287  -1.219    0.223
## poly(thalach, 1) -17.4901     2.6155  -6.687 2.28e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 351.97  on 295  degrees of freedom
## AIC: 355.97
##
## Number of Fisher Scoring iterations: 4
```

```
# prediction and confusion mtx
prob3 <- predict(object = fit3, type = "response")
```

```r
pred3_resp  <- factor(ifelse(prob3 > 0.5, 1, 0), labels = c("0", "1"))
confusionMatrix(pred3_resp, df[,"num"])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 126  52
##          1  34  85
##
##                Accuracy : 0.7104
##                  95% CI : (0.6552, 0.7614)
##     No Information Rate : 0.5387
##     P-Value [Acc > NIR] : 1.042e-09
##
##                   Kappa : 0.4118
##
##  Mcnemar's Test P-Value : 0.06678
##
##             Sensitivity : 0.7875
##             Specificity : 0.6204
##          Pos Pred Value : 0.7079
##          Neg Pred Value : 0.7143
##              Prevalence : 0.5387
##          Detection Rate : 0.4242
##    Detection Prevalence : 0.5993
##       Balanced Accuracy : 0.7040
##
##        'Positive' Class : 0
##
```

We will test our model on test data

```r
sample <- sample.int(n = nrow(df_glm), size = floor(.75*nrow(df_glm)), replace = F)
train_glm <- df_glm[sample, ]
test_glm <- df_glm[-sample, ]
```

```r
test_prob  <- predict(fit3, newdata = test_glm, type = "response")
test_pred_resp  <- factor(ifelse(test_prob > 0.50, 1, 0), labels = c("0", "1"))
confusionMatrix(test_pred_resp, test[,"num"])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 32 17
##          1 16 10
##
##                Accuracy : 0.56
##                  95% CI : (0.4406, 0.6745)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.9395
```

```
##
##                   Kappa : 0.0373
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.6667
##             Specificity : 0.3704
##          Pos Pred Value : 0.6531
##          Neg Pred Value : 0.3846
##              Prevalence : 0.6400
##          Detection Rate : 0.4267
##    Detection Prevalence : 0.6533
##       Balanced Accuracy : 0.5185
##
##        'Positive' Class : 0
##
```

If we try degree 2 or 3, the model will have almost the same prediction power

```r
fit4 <- glm(num ~ poly(thalach, 4), df_glm, family = 'binomial')
summary(fit4)
```

```
##
## Call:
## glm(formula = num ~ poly(thalach, 4), family = "binomial", data = df_glm)
##
## Coefficients:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -0.17027    0.13504  -1.261    0.207
## poly(thalach, 4)1 -17.81129    3.15124  -5.652 1.58e-08 ***
## poly(thalach, 4)2  -0.04233    4.17703  -0.010    0.992
## poly(thalach, 4)3  -1.42777    4.67367  -0.305    0.760
## poly(thalach, 4)4   3.66180    4.11709   0.889    0.374
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 350.58  on 292  degrees of freedom
## AIC: 360.58
##
## Number of Fisher Scoring iterations: 5
```

```r
# prediction and confusion mtx
test_prob4  <- predict(fit4, newdata = test_glm, type = "response")
test_pred_resp4  <- factor(ifelse(test_prob4 > 0.50, 1, 0), labels = c("0", "1"))
confusionMatrix(test_pred_resp4, test[,"num"])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 31 16
##          1 17 11
##
##                Accuracy : 0.56
##                  95% CI : (0.4406, 0.6745)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.9395
##
##                   Kappa : 0.0528
##
##  Mcnemar's Test P-Value : 1.0000
##
##             Sensitivity : 0.6458
##             Specificity : 0.4074
##          Pos Pred Value : 0.6596
##          Neg Pred Value : 0.3929
##              Prevalence : 0.6400
##          Detection Rate : 0.4133
##    Detection Prevalence : 0.6267
##       Balanced Accuracy : 0.5266
##
##        'Positive' Class : 0
##
```

We also can check the full model with adjusted cutoff based on ROCR curve

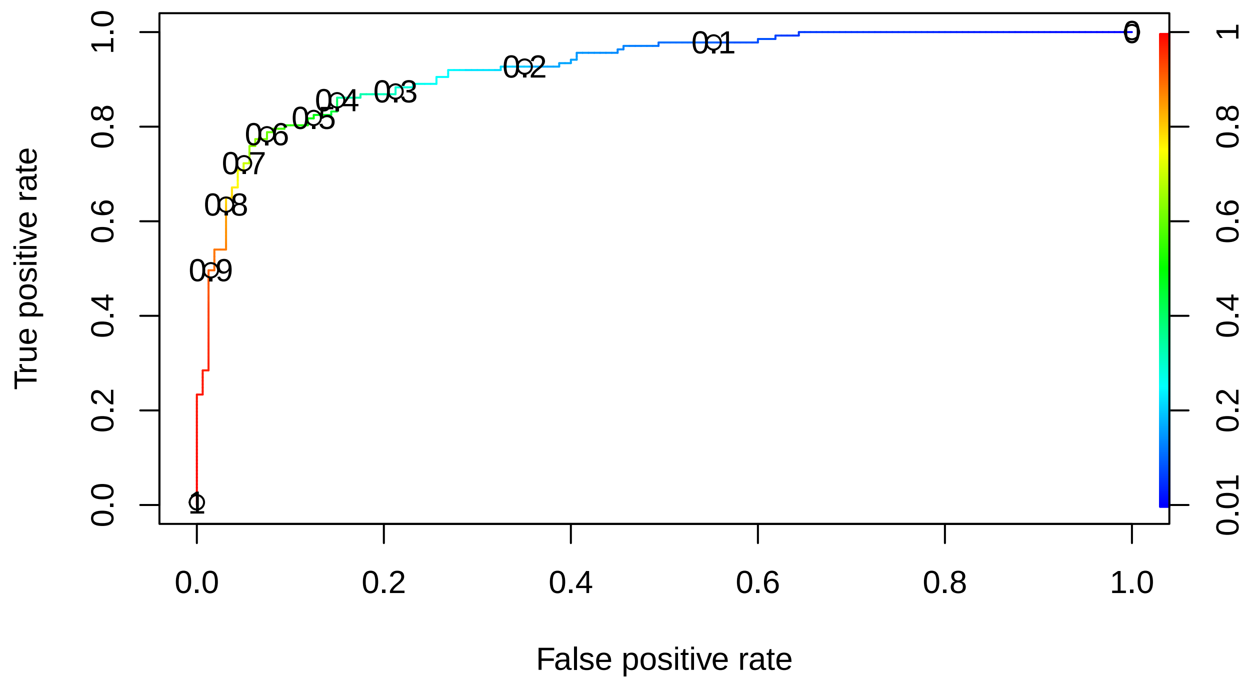
```
fit <- glm(num ~ . , df_glm, family = 'binomial')
summary(fit)
```

```
##
## Call:
## glm(formula = num ~ ., family = "binomial", data = df_glm)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.372042   2.879476  -2.560  0.01046 *
## age         -0.014164   0.023970  -0.591  0.55459
## sex          1.312073   0.488474   2.686  0.00723 **
## cp           0.575898   0.191197   3.012  0.00259 **
## trestbps     0.024044   0.010730   2.241  0.02504 *
## chol         0.004995   0.003774   1.324  0.18561
## fbs         -1.021918   0.555330  -1.840  0.06574 .
## restecg      0.245153   0.185005   1.325  0.18513
## thalach     -0.020665   0.010225  -2.021  0.04327 *
## exang        0.926104   0.413343   2.241  0.02506 *
## oldpeak      0.247386   0.211832   1.168  0.24287
## slope        0.570009   0.363085   1.570  0.11644
## ca           1.267719   0.265384   4.777 1.78e-06 ***
## thal         0.343936   0.100361   3.427  0.00061 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
```

```
##
##     Null deviance: 409.95  on 296  degrees of freedom
## Residual deviance: 204.69  on 283  degrees of freedom
## AIC: 232.69
##
## Number of Fisher Scoring iterations: 6
```

```
prob <- predict(object = fit, type = "response")
```

```
pred_fit <- prediction(prob, df_glm$num)
# true-pos rate, false-pos rate
perf_fit <- performance(pred_fit,"tpr","fpr")
plot(perf_fit, colorize=T , print.cutoffs.at = seq(0,1,by=0.1))
```

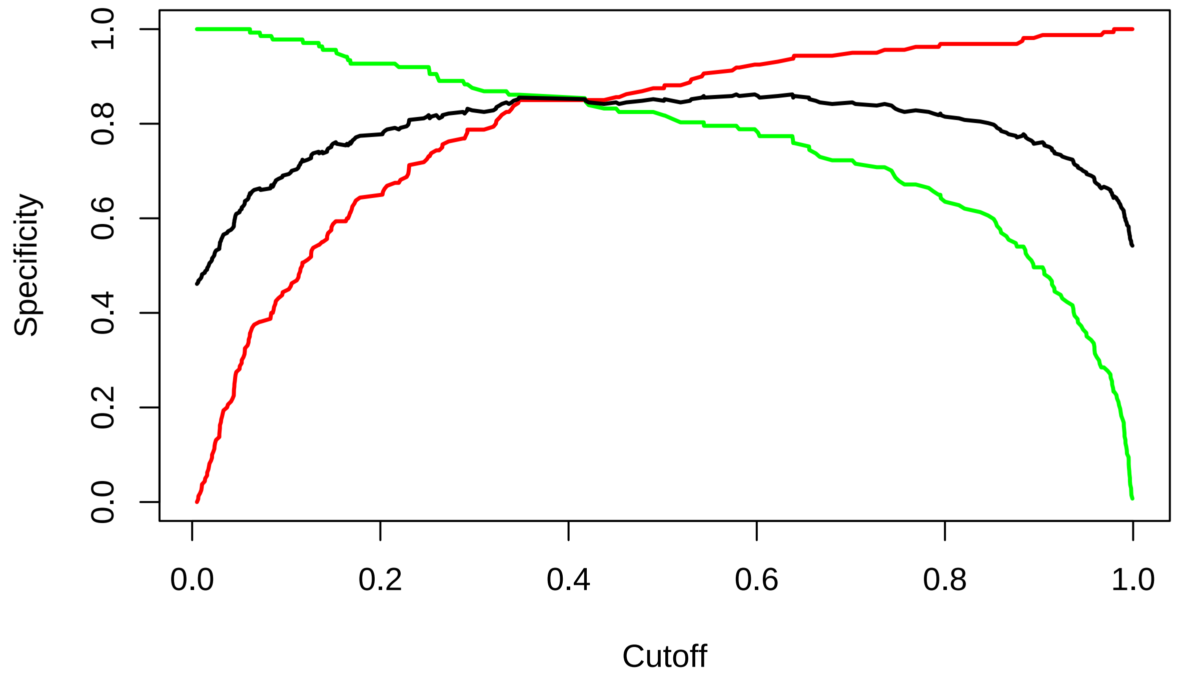


```
auc <- performance(pred_fit, measure = 'auc')

# S under the curve
auc@y.values
```

```
## [[1]]
## [1] 0.924635
```

```r
# the best cutoff will be in the intersection of accuracy, sensitivity and specificity
perf3  <- performance(pred_fit, x.measure = "cutoff", measure = "spec")
perf4  <- performance(pred_fit, x.measure = "cutoff", measure = "sens")
perf5  <- performance(pred_fit, x.measure = "cutoff", measure = "acc")
plot(perf3, col = "red", lwd =2)
plot(add=T, perf4 , col = "green", lwd =2)
plot(add=T, perf5, lwd =2)
```



```r
# based on the plots we choose 0.4 cutoff
pred_resp  <- factor(ifelse(prob > 0.4, 1, 0), labels = c("0", "1"))
confusionMatrix(pred_resp, df[,"num"])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 136  20
##          1  24 117
##
##                Accuracy : 0.8519
##                  95% CI : (0.8063, 0.8902)
##     No Information Rate : 0.5387
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7025
```

```
##
##  Mcnemar's Test P-Value : 0.6511
##
##             Sensitivity : 0.8500
##             Specificity : 0.8540
##          Pos Pred Value : 0.8718
##          Neg Pred Value : 0.8298
##              Prevalence : 0.5387
##          Detection Rate : 0.4579
##    Detection Prevalence : 0.5253
##       Balanced Accuracy : 0.8520
##
##        'Positive' Class : 0
##
```

```
test_prob  <- predict(fit, newdata = test_glm, type = "response")
test_pred_resp  <- factor(ifelse(test_prob > 0.4, 1, 0), labels = c("0", "1"))
confusionMatrix(test_pred_resp, test[,"num"])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 29 10
##          1 19 17
##
##                Accuracy : 0.6133
##                  95% CI : (0.4938, 0.7236)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.7285
##
##                   Kappa : 0.2179
##
##  Mcnemar's Test P-Value : 0.1374
##
##             Sensitivity : 0.6042
##             Specificity : 0.6296
##          Pos Pred Value : 0.7436
##          Neg Pred Value : 0.4722
##              Prevalence : 0.6400
##          Detection Rate : 0.3867
##    Detection Prevalence : 0.5200
##       Balanced Accuracy : 0.6169
##
##        'Positive' Class : 0
##
```

MSE (Mean Squared Error) represents the difference between the original and predicted values extracted by squared the average difference over the data set.