Assignment 6

Ex 1 CORIS data

```
coris <- read.table("coris.dat", skip = 4, sep = ",",
                    col.names = c("row.names",
                                  "sbp",
                                  "tobacco",
                                  "ldl",
                                  "adiposity",
                                  "famhist",
                                  "typea",
                                  "obesity",
                                  "alcohol",
                                  "age",
                                  "chd"))[,-1] # take all the columns exept the first
coris$chd <- as.factor(coris$chd) # Change the response as factor
```

Ex 1.1

Fit the full logistic regression model using all the predictors. Obtain estimations of the accuracy using both leave-one-out and 10-fold cross validation.

```
full_model <- glm(chd ~ ., family ="binomial", data = coris)
summary(full_model)
```

```
##
## Call:
## glm(formula = chd ~ ., family = "binomial", data = coris)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7781  -0.8213  -0.4387   0.8889   2.5435
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -6.1507209  1.3082600  -4.701 2.58e-06 ***
## sbp          0.0065040  0.0057304   1.135 0.256374
## tobacco      0.0793764  0.0266028   2.984 0.002847 **
## ldl          0.1739239  0.0596617   2.915 0.003555 **
## adiposity    0.0185866  0.0292894   0.635 0.525700
## famhist      0.9253704  0.2278940   4.061 4.90e-05 ***
## typea        0.0395950  0.0123202   3.214 0.001310 **
## obesity     -0.0629099  0.0442477  -1.422 0.155095
## alcohol      0.0001217  0.0044832   0.027 0.978350
## age          0.0452253  0.0121298   3.728 0.000193 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 596.11  on 461  degrees of freedom
## Residual deviance: 472.14  on 452  degrees of freedom
## AIC: 492.14
##
## Number of Fisher Scoring iterations: 5
```

For all Gherardo solutions I need a function to transfrom the prediction output to the class values, it takes a prediction, the levels of the response variable, the inverse of the link function of the logistic regression

```
toClass <- function(predictions, levels, linkinv = binomial()$linkinv){
  ## threshold the prob of success
  a <- linkinv(predictions) > 0.5  # if prob succ > 0.5 => success
  b <- array(dim = c(length(predictions)))

  b[a] <- levels[2] # (second lvl)
  b[!a] <- levels[1] # otherwise not success (first lvl)

return(factor(b, levels = levels)) # we should return a factor
}

predicted <- toClass(predict(full_model), levels(coris$chd))
tt <- table(predicted, coris$chd)
acc_train <- sum(diag(tt)) / sum(tt) # accuracy the training set
acc_train
```

```
## [1] 0.7337662
```

Cross validation leave 1 out

```
# Check that the inverse of the link function > 0.5 OR P(chd) > 0.5
rcv_calc <- function(model = chd  ~ .){
correct_pred = 0              # correct prediction = T positive + T negative
  for(i in 1:nrow(coris)) {
  # fit model with all but i row
    rcv_model <- glm(formula = model, family ="binomial", data = coris[-i,])
  # predict value for row i
    chd_pred <- as.numeric(predict(rcv_model, newdata = coris[i,], type = "response") > 0.5)    # predic
  # check if prediction is good or not and count the good predictions
    correct_pred <- correct_pred + (coris[i,]$chd == chd_pred)                                  # correc
  }
  # save it calculate average
  return(correct_pred/nrow(coris))
}
rcv_calc()                        # accuracy full model leave 1 out
```

```
## [1] 0.7186147
```

```
  # or from Gherardo solutions
n <- nrow(coris)
res_loo <- sapply(1:n, function(i){
  fit <- glm(chd ~ ., data = coris[-i, ], family = "binomial")
  pr <- predict(fit, newdata = coris[i,])
  pred.class <- toClass(pr, levels = levels(coris$chd))
```

```
  return(pred.class == coris$chd[i])
})
acc_loo <- sum(res_loo) / length(res_loo)
acc_loo
```

```
## [1] 0.7186147
```

K-fold cross validation

```
kfold_rcv_calc <- function(key = 5, model = chd  ~ ., shuffle = TRUE){
  if (shuffle){
    n <-  nrow(coris)                          # if you want to generalize just add data = model$data
    coris <- coris[sample(1:n),]               # and change all the names coris to data
  }
  correct_pred = 0
  k <- key
  folds <- list()
  n <- nrow(coris) %/% k

  for (i in 1:k){
    folds[[i]] <- ((i-1) * n + 1):(i * n)

    rcv_model <- glm(formula = model, family ="binomial", data = coris[-folds[[i]],])
    chd_pred <- as.numeric(
      predict(rcv_model, newdata = coris[folds[[i]],], type = "response") > 0.5)
    correct_pred <- correct_pred + sum(coris[folds[[i]],]$chd == chd_pred)
  }
  return(correct_pred/ (k * n))
}
kfold_rcv_calc(10, shuffle = TRUE)
```

```
## [1] 0.7130435
```

```
  # or from Gherardo during lecture
kfold_rcv_calc2 <- function(key = 5, model = chd  ~ .){
  correct_pred = 0
  k <- key
  folds <- list()
  m <- nrow(coris) %/% k
  res <- c()

  for (i in 1:k){
    folds[[i]] <- ((i-1) * m + 1):(i * m)

    rcv_model <- glm(formula = model, family ="binomial", data = coris[-folds[[i]],])
    chd_pred <- as.numeric(
      predict(rcv_model, newdata = coris[folds[[i]],], type = "response") > 0.5)

    comparison <- coris[folds[[i]],]$chd == chd_pred
    res[i] <- sum(comparison)
  }
  return(res)
}
kfold_rcv_calc2(10)
```

```
##  [1] 31 35 33 34 27 31 36 34 37 34
```

```
mean(kfold_rcv_calc2(10)/46)              # accuracy full model k-fold
```

```
## [1] 0.7217391
```

Repeat with shuffled data (I updated the previous function with shuffl) and groups that include all observations

```
# 2) Shuffle the dataset and prepare the groups for the k-fold rcv (more precise groups)
n <- nrow(coris)
corisSHF <- coris[sample(1:n), ]

k <- 10
r <- floor(n / k)
groups <- list()
t <- 0
s <- 0
for (i in 1:10){ # in this way is more complicated but it include all the observations
  if (i > 8){
    t <- 1
    if (i > 9){
      s <- 1
    }
  }
  groups[[i]] <- ((i - 1) * r + 1 + s) : (i * r + t + s)
}

# 3) Cross validation with shuffled data (full model)
res_10 <- sapply(1:10, function(i){
  fit <- glm(chd ~ ., data = corisSHF[-groups[[i]], ], family = "binomial")
  pr <- predict(fit, newdata = corisSHF[groups[[i]],])
  pred.class <- toClass(pr, levels = levels(coris$chd))
  tt <- table(pred.class, corisSHF$chd[ groups[[i]] ] )
  acc <- sum(diag(tt)) / sum(tt)
  return(acc)
})
acc_10 <- mean(res_10)
acc_10
```

```
## [1] 0.7120259
```

Stepwise backward selection

```
step_model <- step(full_model, direction = "backward", trace = 0)

matrix(c(rcv_calc(), kfold_rcv_calc(10), rcv_calc(step_model), kfold_rcv_calc(10, step_model)),
         nrow = 2, byrow = TRUE, dimnames = list(model = c("full" , "step"),
         accuracy = c("loo", "10-fold")) )
```

```
##       accuracy
## model        loo   10-fold
##   full 0.7186147 0.7217391
##   step 0.7359307 0.7413043
```

Repeat with shuffled dataset for step model k-fold cross validation

```
res_10_st <- sapply(1:10, function(i){
  fit <- glm(formula(step_model), data = corisSHF[-groups[[i]], ], family = "binomial")
  pr <- predict(fit, newdata = corisSHF[groups[[i]],])
  pred.class <- toClass(pr, levels = levels(coris$chd))
  tt <- table(pred.class, corisSHF$chd[ groups[[i]] ] )
  acc <- sum(diag(tt)) / sum(tt)
return(acc)
})
acc_10_st <- mean(res_10_st)
acc_10_st
```

```
## [1] 0.740148
```

```
matrix(c(rcv_calc(), acc_10, rcv_calc(step_model), acc_10_st), nrow = 2, byrow = TRUE,
dimnames = list(model = c("full", "step"), accuracy = c("loo     ", "10-fold SHF")))
```

```
##       accuracy
## model   loo      10-fold SHF
##   full 0.7186147   0.7120259
##   step 0.7359307   0.7401480
```

We can observe that the simpler model obtained with backward stepwise selection based on AIC, generalize better, that is has a better accuracy over unseen observations.

Ex 1.2 Implement stepwise forward selection using accuracy estimated with 5fold cross-validation to score the candidate models.

RCV stepwise selection by my kfold function

```
# (Exercise 3.1 there is a function for the step using AIC)
fit <- glm(chd ~ 1, family = binomial, data = coris) # we need a model with only the intercept
regressors <- colnames(coris)[-10] # we select the regressors
selected <- c()
score <- kfold_rcv_calc(5, fit)
score.best <- score
done <- FALSE
# continue to add regressors until there is not decrease in AIC (done = TRUE)
while (!done){
# for regressors in regressors list exept the regressors already used
  for (reg in regressors[!(regressors %in% selected)]){
    # update the fit model adding one regressor each time
    tmp <- update(fit, formula = paste(". ~ . + ", reg))
    # calculate accuracy by Kf-CV
    score.tmp <- kfold_rcv_calc(5, tmp)
    # if accuracy is larger than this is the best score
    if (score.tmp > score.best){
      # change the best score in to this one (store it outside the for loop)
      score.best <- score.tmp
      # store the updated model that just give the best score
      best <- tmp
      # store the selected parameters
      selected.best <- c(selected, reg)
    }
  }
```

```
  # when the for loop finish, if score.best > score
  if (score.best > score){
    fit <- best                # store best model to fit
    score <- score.best        # score best score to score
    selected <- selected.best  # store select models that score
  }else{ # if there is no decrease
  done <- TRUE
  }
}
impl_model <- fit
impl_model
```

```
##
## Call:  glm(formula = chd ~ tobacco + obesity + alcohol, family = binomial,
##     data = coris)
##
## Coefficients:
## (Intercept)      tobacco      obesity      alcohol
##  -2.0488647    0.1409699    0.0333558    0.0001118
##
## Degrees of Freedom: 461 Total (i.e. Null);  458 Residual
## Null Deviance:       596.1
## Residual Deviance: 552.8     AIC: 560.8
```

```
acc_full <- kfold_rcv_calc(5, full_model) # full model
acc_step <- kfold_rcv_calc(5, step_model) # step by built in function
acc_impl <- kfold_rcv_calc(5, impl_model) # my step implemented

data.frame(Accuracy = c(acc_full, acc_step, acc_impl), Models = c("full", "step", "impl"))
```

```
##    Accuracy Models
## 1 0.7173913   full
## 2 0.7239130   step
## 3 0.6913043   impl
```

```
matrix(c(acc_full, acc_step, acc_impl), nrow = 3, byrow = TRUE,
dimnames = list(model = c("full", "step", "impl"), accuracy = "5-fold CV"))
```

```
##       accuracy
## model  5-fold CV
##   full 0.7173913
##   step 0.7239130
##   impl 0.6913043
```

RCV stepwise selection by Gherardo solutions

```
# RCV function that perform leave 1 out by default
crossval <- function(object, data = object$data,
                     groups = as.list(1:nrow(data)),
                     shuffle = TRUE) {
  if (shuffle) {
    data <- data[sample(1:nrow(data)),]
  }
  class <- as.character(object$formula[[2]])
  res <- sapply(groups, function(ix) {
```

```
    modello <- glm(formula(object), data = data[-ix,], family = object$family)
    pr <- predict(modello, newdata = data[ix,])
    pred.class <- toClass(pr, levels = levels(data[[class]]),
                          linkinv = object$family$linkinv)
    tt <- table(pred.class, data[[class]][ix])
    acc <- sum(diag(tt)) / sum(tt)
    return(acc)
  })
  return(mean(res))
}

# Test the function
crossval(full_model)
```

```
## [1] 0.7186147
```

```
crossval(full_model, data = corisSHF, groups = groups, shuffle = FALSE)
```

```
## [1] 0.7120259
```

It seems that it works fine.

```
# Stepwise selection using the RCV      (Exercise 3.1 there is a function for the step using AIC)
k <- 5
r <- floor(n / k)
groups <- list()
t <- 0
s <- 0
for (i in 1:5) {
  if (i > 3) {
    t <- 1
    if (i > 4) {
      s <- 1
    }
  }
  groups[[i]] <- ((i - 1) * r + 1 + s):(i * r + t + s)
}
### we start the model with only the intercept
fit <- glm(chd ~ 1, family = "binomial",
           data = coris) ## only the intercept
regressors <- colnames(coris)[-10]
selected <- c()
score <- crossval(object = fit,
                  groups = groups,
                  shuffle = FALSE)
score.best <- score
done <- FALSE
while (!done) {
  for (reg in regressors[!(regressors %in% selected)]) {
    tmp <- update(fit, formula = paste(". ~ . + ", reg))
    score.tmp <- crossval(tmp, groups = groups, shuffle = FALSE)
    if (score.tmp > score.best) {
      score.best <- score.tmp
      best <- tmp
      selected.best <- c(selected, reg)
```

```
    }
  }
  if (score.best > score) {
    fit <- best
    score <- score.best
    selected <- selected.best
  } else{
    ### if there is no increase
    done <- TRUE
  }
}
impl_model2 <- fit
impl_model2
```

```
##
## Call:  glm(formula = chd ~ tobacco + ldl + famhist + age, family = "binomial",
##     data = coris)
##
## Coefficients:
## (Intercept)      tobacco          ldl      famhist          age
##    -4.20428      0.08070      0.16758      0.92412      0.04404
##
## Degrees of Freedom: 461 Total (i.e. Null);  457 Residual
## Null Deviance:       596.1
## Residual Deviance: 485.4     AIC: 495.4
```

```
acc_step <- kfold_rcv_calc(5, step_model)
acc_impl <- kfold_rcv_calc(5, impl_model)
acc_cv1 <- crossval(full_model, groups = groups, shuffle = FALSE) # full model
acc_cv2 <- crossval(step_model, groups = groups, shuffle = FALSE) # step by built in function
acc_cv3 <- crossval(impl_model, groups = groups, shuffle = FALSE) # my step implemented
acc_cv4 <- crossval(impl_model2, groups = groups, shuffle = FALSE)# Ghe step implemented (use all obs)

data.frame(Accuracy = c(acc_cv1, acc_cv2, acc_cv3, acc_cv4), Models = c("full", "step", "my impl1", "gh
```

```
##    Accuracy    Models
## 1 0.7270220      full
## 2 0.7292660      step
## 3 0.6946470  my impl1
## 4 0.7335671 ghe impl2
```

Ex 2 The wine quality dataset

We load both red and white wine datasets and we transform the quality index to a binary good-bad variable.

```
wines_red <- read.csv("winequality-red.csv", sep =";")
wines_white <- read.csv("winequality-white.csv", sep =";")

good <- wines_red$quality > 5
wines_red$quality <- "bad"
wines_red[good, "quality"] <- "good"
wines_red[,"quality"] <- as.factor(wines_red[, "quality"])

good <- wines_white$quality > 5
```

```
wines_white$quality <- "bad"
wines_white[good, "quality"] <- "good"
wines_white[,"quality"] <- as.factor(wines_white[, "quality"])
```

Ex 2.1 Fit a logistic regression models using all the predictors and the data for the red wines. Compute the accuracy of the model on the red wines and on the white wines.

```
redfull <- glm(quality ~ ., family = binomial, data = wines_red)
whitefull <- glm(quality ~ ., family = binomial, data = wines_white)

pred_rf <- predict(redfull, newdata = wines_red) # exponent of the linkinv
pred_wf <- predict(whitefull, newdata = wines_white)
pred_rf_linkinv <- predict(redfull, newdata = wines_red, type = "response") # linkinv

##### CLASSIFIERS METHODS #####

### Method 1)        -> with this setting use exp of the linkinv
toClass <- function(predictions, levels, linkinv = binomial()$linkinv){
  a <- linkinv(predictions) > 0.5  # if prob succ > 0.5 => success
  b <- array(dim = c(length(predictions)))
  b[a] <- levels[2] # (second lvl)
  b[!a] <- levels[1] # otherwise not success (first lvl)
return(factor(b, levels = levels)) # we should return a factor
}

predclass1_r <- toClass(pred_rf, levels(wines_red$quality))
predclass1_w <- toClass(pred_wf, levels(wines_white$quality))

confusion1_r <- table(predclass1_r, wines_red$quality)
confusion1_w <- table(predclass1_w, wines_white$quality)

acc_red <- sum(diag(confusion1_r)) / sum(confusion1_r)  # accuracy the training set
paste("accuracy red on training set:  ", acc_red)
```

```
## [1] "accuracy red on training set:   0.744215134459037"
```

```
acc_white <- sum(diag(confusion1_w)) / sum(confusion1_w)
paste("accuracy white on training set:", acc_white)
```

```
## [1] "accuracy white on training set: 0.750102082482646"
```

```
### Method 2)         -> atm use linkinv
predclass2 <- factor(sapply(pred_rf_linkinv, function(x)
    ifelse(x < 0.5, "Bad", "Good")), levels = c("Bad", "Good"))
confusion2 <- table(predclass2, wines_red$quality)

errs <- sum(confusion2) - sum(diag(confusion2))
accuracy <- 1 - errs/sum(confusion2)
accuracy
```

```
## [1] 0.7442151
```

```
accuracy <- sum(diag(confusion2)) / sum(confusion2)
accuracy
```

```
## [1] 0.7442151
```

```
### Method 3)          -> works only with binary 0 1, atm use linkinv
wines_red[,"quality"] <- as.numeric(wines_red[, "quality"])
wines_red$quality[wines_red$quality == 1] <- 0
wines_red$quality[wines_red$quality == 2] <- 1

redfull <- glm(quality ~ ., family = binomial, data = wines_red)
pred_rf_linkinv <- predict(redfull, newdata = wines_red, type = "response")

pred <- as.numeric(pred_rf_linkinv > 0.5)

# Accuracy calculation 1
confusion <- table(pred, wines_red$quality)
accuracy <- sum(diag(confusion2)) / sum(confusion2)
accuracy
```

```
## [1] 0.7442151
```

```
# Accuracy calculation 2
correct_pred <- pred == wines_red$quality
sum(correct_pred)/nrow(wines_red)
```

```
## [1] 0.7442151
```

```
## Change back to factor
wines_red[,"quality"] <- as.factor(wines_red[, "quality"])
levels(wines_red$quality) <- c("bad", "good")
levels(wines_red$quality)
```

```
## [1] "bad"  "good"
```

All of the previous method can work with both linkinv or exponent of the linkinv

I repeat the measurement with k-fold CV

```
#### k-fold ALL INCLUSIVE 2 functions

# Calculate groups
calc_groups <- function(data, key = 10) {
  k <- key
  groups <- list()
  m <- nrow(data) %/% k
  for (i in 1:k){
    groups[[i]] <- ((i-1) * m + 1):(i * m)
  }
  return(groups)
}

# Adapted k-fold crossval
crossval <- function(object,
                     data = object$data,
                     groups = as.list(1:nrow(data)),
                     kfold = FALSE,
                     key = 10,
                     shuffle = TRUE) {
```

```
  if (kfold) {
    groups = calc_groups(data, key = 10)
  }
  if (shuffle) {
    data <- data[sample(1:nrow(data)),]
  }
  class <- as.character(object$formula[[2]])
  res <- sapply(groups, function(ix) {
    modello <- glm(formula(object), data = data[-ix,], family = object$family)
    pr <- predict(modello, newdata = data[ix,])
    pred.class <- toClass(pr, levels = levels(data[[class]]),
                          linkinv = object$family$linkinv)
    tt <- table(pred.class, data[[class]][ix])
    acc <- sum(diag(tt)) / sum(tt)
    return(acc)
  })
  return(mean(res))
}
# perform leave 1 out (shuffle has no effect)
crossval(object = redfull, data = wines_red)
```

```
## [1] 0.7398374
```

```
# perform k-fold, can specify fold size
crossval(object = redfull, data = wines_red, kfold = TRUE, key = 5)
```

```
## [1] 0.7465409
```

```
# perform k-fold, use the same group
crossval(object = redfull, data = wines_red, kfold = TRUE, shuffle = FALSE)
```

```
## [1] 0.7408805
```

```
## Calculate accuracy
crossval(object = redfull, data = wines_red, kfold = TRUE)
```

```
## [1] 0.7433962
```

```
crossval(object = whitefull, data = wines_white, kfold = TRUE)
```

```
## [1] 0.7490798
```

2.2 Fit a logistic regression model using the white wines data and compute the accuracy over the white and red wines.

```
# Function to calculate accuracy
calc_acc <- function(object, data = object$data){
  pred <- predict(object, newdata = data) # exponent of the linkinv
  pred.class <- toClass(pred, levels(data$quality))
  confusion <- table(pred.class, data$quality)
  acc <- sum(diag(confusion)) / sum(confusion)
  return(acc)
}

acc_r.w_full <- calc_acc(redfull, wines_white)
```

```r
paste("accuracy red model on white dataset:", acc_r.w_full)
```

```
## [1] "accuracy red model on white dataset: 0.667211106574112"
```

```r
acc_w.r_full <- calc_acc(whitefull, wines_red)
paste("accuracy white model on red dataset:", acc_w.r_full)
```

```
## [1] "accuracy white model on red dataset: 0.623514696685428"
```

```r
# cross validation daens't make sense if I'm not using the model on the dataset that I use to train it
```

```r
summary(redfull)
```

```
##
## Call:
## glm(formula = quality ~ ., family = binomial, data = wines_red)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.4025  -0.8387   0.3105   0.8300   2.3142
##
## Coefficients:
##                        Estimate Std. Error z value Pr(>|z|)
## (Intercept)           42.949948  79.473979   0.540  0.58890
## fixed.acidity          0.135980   0.098483   1.381  0.16736
## volatile.acidity      -3.281694   0.488214  -6.722 1.79e-11 ***
## citric.acid           -1.274347   0.562730  -2.265  0.02354 *
## residual.sugar         0.055326   0.053770   1.029  0.30351
## chlorides             -3.915713   1.569298  -2.495  0.01259 *
## free.sulfur.dioxide    0.022220   0.008236   2.698  0.00698 **
## total.sulfur.dioxide  -0.016394   0.002882  -5.688 1.29e-08 ***
## density              -50.932385  81.148745  -0.628  0.53024
## pH                    -0.380608   0.720203  -0.528  0.59717
## sulphates              2.795107   0.452184   6.181 6.36e-10 ***
## alcohol                0.866822   0.104190   8.320  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2209.0  on 1598  degrees of freedom
## Residual deviance: 1655.6  on 1587  degrees of freedom
## AIC: 1679.6
##
## Number of Fisher Scoring iterations: 4
```

```r
summary(whitefull)
```

```
##
## Call:
## glm(formula = quality ~ ., family = binomial, data = wines_white)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.1731  -0.8946   0.4420   0.7994   2.9466
##
```

```
## Coefficients:
##                         Estimate Std. Error z value Pr(>|z|)
## (Intercept)           2.582e+02  7.099e+01   3.638 0.000275 ***
## fixed.acidity         3.648e-02  7.178e-02   0.508 0.611271
## volatile.acidity     -6.459e+00  4.128e-01 -15.646  < 2e-16 ***
## citric.acid           1.158e-01  3.029e-01   0.382 0.702219
## residual.sugar        1.701e-01  2.704e-02   6.291 3.16e-10 ***
## chlorides             8.852e-01  1.671e+00   0.530 0.596379
## free.sulfur.dioxide   9.601e-03  2.782e-03   3.451 0.000560 ***
## total.sulfur.dioxide -1.333e-03  1.211e-03  -1.101 0.270982
## density              -2.709e+02  7.195e+01  -3.765 0.000167 ***
## pH                    1.090e+00  3.618e-01   3.013 0.002590 **
## sulphates             1.797e+00  3.595e-01   5.000 5.75e-07 ***
## alcohol               7.429e-01  9.361e-02   7.937 2.08e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 6245.4  on 4897  degrees of freedom
## Residual deviance: 4932.6  on 4886  degrees of freedom
## AIC: 4956.6
##
## Number of Fisher Scoring iterations: 5
```

It could be usefull since they have some relevant covariates in common, but some other covariates that are important for the red wines, are irrelevant for the white wines.

Exercise 3

3.1 Use now both red and white wines data and perform stepwise forward selection with AIC to select a logistic regression model for the binary quality variable.

```r
# Stepwise selection using the AIC
step_selection.aic <- function(data){
  fit <- glm(quality ~ 1, family = "binomial",
             data = data) ## only the intercept
  regressors <- colnames(data)[-12]
  selected <- c()
  score <- AIC(fit)
  score.best <- score
  done <- FALSE
  while (!done) {
    for (reg in regressors[!(regressors %in% selected)]) {
      tmp <- update(fit, formula = paste(". ~ . + ", reg))
      score.tmp <- AIC(tmp)
      if (score.tmp < score.best) {
        score.best <- score.tmp
        best <- tmp
        selected.best <- c(selected, reg)
      }
    }
```

```
    if (score.best < score) {
      fit <- best
      score <- score.best
      selected <- selected.best
    } else{
      ### if there is no increase
      done <- TRUE
    }
  }
  return(fit)
}
red_step <- step_selection.aic(wines_red)
red_step
```

```
##
## Call:  glm(formula = quality ~ alcohol + volatile.acidity + total.sulfur.dioxide +
##     sulphates + chlorides + free.sulfur.dioxide + pH + citric.acid +
##     fixed.acidity, family = "binomial", data = data)
##
## Coefficients:
##          (Intercept)              alcohol      volatile.acidity
##              -6.93847              0.91774              -3.30102
## total.sulfur.dioxide            sulphates            chlorides
##              -0.01622              2.70071              -3.94020
##  free.sulfur.dioxide                   pH          citric.acid
##               0.02317             -0.63567              -1.24344
##        fixed.acidity
##               0.09000
##
## Degrees of Freedom: 1598 Total (i.e. Null);  1589 Residual
## Null Deviance:       2209
## Residual Deviance: 1657  AIC: 1677
```

```
white_step <- step_selection.aic(wines_white)
white_step
```

```
##
## Call:  glm(formula = quality ~ alcohol + volatile.acidity + residual.sugar +
##     fixed.acidity + sulphates + free.sulfur.dioxide + density +
##     pH, family = "binomial", data = data)
##
## Coefficients:
##         (Intercept)              alcohol     volatile.acidity
##           2.675e+02            7.371e-01           -6.549e+00
##      residual.sugar        fixed.acidity            sulphates
##           1.715e-01            3.896e-02            1.757e+00
## free.sulfur.dioxide              density                   pH
##           7.924e-03           -2.801e+02            1.066e+00
##
## Degrees of Freedom: 4897 Total (i.e. Null);  4889 Residual
## Null Deviance:       6245
## Residual Deviance: 4934  AIC: 4952
```

Ex 3.2 Estimate the accuracy of the model using 10-fold cross validation on the red and white wines data.

```
# The function perform kfold and can test the model to a different dataset
# -> it is the same of crossval but daesn't perform leave 1 out
kfold_10_step <- function(object, data = object$data, shuffle = TRUE, groups = 10){
  if (shuffle){
    data = data[sample(1:nrow(data)),]
  }
  groups = calc_groups(data, groups)
  acc_10 <- sapply(1:10, function(i){
    fit <- glm(formula(object),
               data = data[-groups[[i]], ], family = "binomial")
    pr <- predict(fit, newdata = data[groups[[i]],])
    pred.class <- toClass(pr, levels = levels(data$quality))
    tt <- table(pred.class, data$quality[ groups[[i]] ] )
    acc <- sum(diag(tt)) / sum(tt)
    return(acc)
  })
  return(mean(acc_10))
}

kfold_10_step(red_step, wines_red)
```

```
## [1] 0.7402516
```

```
kfold_10_step(white_step, wines_white)
```

```
## [1] 0.7505112
```

```
kfold_10_step(red_step, wines_white)
```

```
## [1] 0.7323108
```

```
kfold_10_step(white_step, wines_red)
```

```
## [1] 0.7251572
```

Ex 3.3 We want now to estimate the accuracy of the models trained both on red and white wines only over the red wines

```
# Function that take a dataset return a training set and a target set
train_target_gen <- function(data, n){
  i <- sample(1:1599, size = n, replace = FALSE)
  train <- data[-i,]
  target <- data[i,]
  return(list(train = train, target = target))
}

accuracy_200 <- function(){
  i <- sample(1:1599, size = 200, replace = FALSE)

  red_train <- wines_red[-i,]
  white_train <- wines_white[-i,]
  all_train <- rbind(red_train, white_train)

  red_target <- wines_red[i,]
```

```
  red_step_rnd <- step_selection.aic(all_train)
  acc_red <- calc_acc(object = red_step_rnd, data = all_train)

  print("..loading..")
  return(acc_red)
}

acc_rnd <- replicate(20, accuracy_200())
```

```
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
## [1] "..loading.."
```

```
acc_rnd_mean <- mean(acc_rnd)
acc_rnd_mean
```

```
## [1] 0.7423733
```