Modelling CPU usage by combining a classification step and a regression step

Álvaro Francesc Budría Fernández

Introduction

This document presents my analysis of a dataset containing data about a computer's usage. The final aim is to model the usage of a computer's CPU through a regression model. There are 8192 observations with 21 features plus a target. The approach taken here is to first classify observations as either active or inactive. Then inactive CPUs are predicted to have an activity of 0%, while the active CPUs activity is predicted with a regression model. Several models are compared through a 10fold CV procedure.

```
library(MASS) # LDA, QDA
library(class) # kNN
library(TunePareto) # for generateCVRuns()
library(glmnet) # ridge regression
library(nnet) # MLP
library(caret) # train MLPNN
library(RSNNS)
```

Read data

```
X <- read.csv('cpu.csv')
```

Check for missing values, anomalies, possible errors…

```
sum(is.na(X))
```

```
## [1] 0
```

```
summary(X)
```

```
##      lread            lwrite           scall           sread
##  Min.   :   0.00   Min.   :  0.00   Min.   :  109   Min.   :   6.0
##  1st Qu.:   2.00   1st Qu.:  0.00   1st Qu.: 1012   1st Qu.:  86.0
##  Median :   7.00   Median :  1.00   Median : 2052   Median : 166.0
##  Mean   :  19.56   Mean   : 13.11   Mean   : 2306   Mean   : 210.5
##  3rd Qu.:  20.00   3rd Qu.: 10.00   3rd Qu.: 3317   3rd Qu.: 279.0
##  Max.   :1845.00   Max.   :575.00   Max.   :12493   Max.   :5318.0
##      swrite           fork             exec            rchar
##  Min.   :   7.0   Min.   : 0.000   Min.   : 0.000   Min.   :    278
##  1st Qu.:  63.0   1st Qu.: 0.400   1st Qu.: 0.200   1st Qu.:  33864
##  Median : 117.0   Median : 0.800   Median : 1.200   Median : 124780
##  Mean   : 150.1   Mean   : 1.885   Mean   : 2.792   Mean   : 197014
##  3rd Qu.: 185.0   3rd Qu.: 2.200   3rd Qu.: 2.800   3rd Qu.: 267669
##  Max.   :5456.0   Max.   :20.120   Max.   :59.560   Max.   :2526649
```

```
##      wchar              pgout             ppgout             pgfree       
##  Min.   :   1498   Min.   : 0.000   Min.   :  0.000   Min.   :  0.00  
##  1st Qu.:  22936   1st Qu.: 0.000   1st Qu.:  0.000   1st Qu.:  0.00  
##  Median :  46620   Median : 0.000   Median :  0.000   Median :  0.00  
##  Mean   :  95898   Mean   : 2.285   Mean   :  5.977   Mean   : 11.92  
##  3rd Qu.: 106148   3rd Qu.: 2.400   3rd Qu.:  4.200   3rd Qu.:  5.00  
##  Max.   :1801623   Max.   :81.440   Max.   :184.200   Max.   :523.00  
##      pgscan             atch               pgin              ppgin       
##  Min.   :   0.00   Min.   :  0.000   Min.   :  0.000   Min.   :  0.00  
##  1st Qu.:   0.00   1st Qu.:  0.000   1st Qu.:  0.600   1st Qu.:  0.60  
##  Median :   0.00   Median :  0.000   Median :  2.800   Median :  3.80  
##  Mean   :  21.53   Mean   :  1.127   Mean   :  8.278   Mean   : 12.39  
##  3rd Qu.:   0.00   3rd Qu.:  0.600   3rd Qu.:  9.765   3rd Qu.: 13.80  
##  Max.   :1237.00   Max.   :211.580   Max.   :141.200   Max.   :292.61  
##       pflt             vflt             runqsz            freemem     
##  Min.   :  0.0   Min.   :   0.2   Min.   :   1.00   Min.   :   55  
##  1st Qu.: 25.0   1st Qu.:  45.4   1st Qu.:   1.20   1st Qu.:  231  
##  Median : 63.8   Median : 120.4   Median :   2.00   Median :  579  
##  Mean   :109.8   Mean   : 185.3   Mean   :  19.63   Mean   : 1763  
##  3rd Qu.:159.6   3rd Qu.: 251.8   3rd Qu.:   3.00   3rd Qu.: 2002  
##  Max.   :899.8   Max.   :1365.0   Max.   :2823.00   Max.   :12027  
##     freeswap            usr       
##  Min.   :      2   Min.   : 0.00  
##  1st Qu.:1042624   1st Qu.:81.00  
##  Median :1289290   Median :89.00  
##  Mean   :1328126   Mean   :83.97  
##  3rd Qu.:1730380   3rd Qu.:94.00  
##  Max.   :2243187   Max.   :99.00  
```

```
rbind(apply(X, 2, mean), apply(X, 2, sd))
```

```
##         lread   lwrite    scall    sread   swrite     fork     exec    rchar
## [1,] 19.55969 13.10620 2306.318 210.4800 150.0582 1.884554 2.791998 197013.7
## [2,] 53.35380 29.89173 1633.617 198.9801 160.4790 2.479493 5.212456 239480.8
##          wchar    pgout    ppgout   pgfree   pgscan     atch     pgin    ppgin
## [1,]  95898.29 2.285317  5.977229 11.91971 21.52685 1.127505  8.27796 12.38859
## [2,] 140756.86 5.307038 15.214590 32.36352 71.14134 5.708347 13.87498 22.28132
##          pflt     vflt    runqsz  freemem  freeswap      usr
## [1,] 109.7938 185.3158  19.63068 1763.456 1328126.0 83.96887
## [2,] 114.4192 191.0006 125.74209 2482.105  422019.4 18.40190
```

```
# for (i in 1:22) # commented because output is huge
  # boxplot(X[,i], main=colnames(X)[i])
```

No missing values codified as NA. There are no other common codifications for NAs, such as -1, -99… so we conclude that there are no missing data in the dataset. The boxplots allow us to detect many univariate outliers.

We can see quite a lot of outliers for all the variables. If we look further into the observations that are showing an outlier for a particular variable, we can see that those observations are not necessarily outliers for other variables. Thus, it is not wise to remove observations with at least one outlier variable, as this would result in too many lost data.

Compute correlation between variables

```r
mosthighlycorrelated <- function(mydataframe,numtoreport)
{
  # find the correlations
  cormatrix = cor(mydataframe)
  # set the correlations on the diagonal or lower triangle to zero,
  # so they will not be reported as the highest ones:
  diag(cormatrix) = 0
  cormatrix[lower.tri(cormatrix)] = 0
  # flatten the matrix into a dataframe for easy sorting
  fm = as.data.frame(as.table(cormatrix))
  # assign human-friendly names
  names(fm) = c("First.Variable", "Second.Variable","Correlation")
  # sort and print the top n correlations
  head(fm[order(abs(fm$Correlation),decreasing=TRUE),],n=numtoreport)
}

mosthighlycorrelated(X, 15)
```

```
##     First.Variable Second.Variable Correlation
## 380           fork            vflt   0.9393485
## 391           pflt            vflt   0.9353696
## 358           fork            pflt   0.9310400
## 345           pgin           ppgin   0.9236207
## 253         ppgout          pgfree   0.9177905
## 276         pgfree          pgscan   0.9152168
## 92           sread          swrite   0.8810694
## 230          pgout          ppgout   0.8724454
## 275         ppgout          pgscan   0.7852563
## 138           fork            exec   0.7639742
## 252          pgout          pgfree   0.7303810
## 69           scall           sread   0.6968868
## 381           exec            vflt   0.6917545
## 483       freeswap             usr   0.6785262
## 359           exec            pflt   0.6452390
```

We eliminate variables fork, pflt, ppgout, ppgin, pgscan, sread, as they are highly correlated with other variables, and therefore redundant.

```r
Xclean <- subset(X, select = -c(fork, pflt, ppgout, ppgin, pgscan, sread) )
```

Split data into training (70%) and testing (30%) sets

```r
set.seed(12345)

N <- nrow(X)
train <- sample(1:N, round(2*N/3))
ntrain <- length(train)
ntest <- N - ntrain

Xtrain <- Xclean[train,]
Xtest <- Xclean[-train,]
```

There is something odd in the data. The dynamic range of the target usr seems to be split into two subintervals:

one going from 0 to 4, the other from 4 to 100. Let's verify this hypothesis by running a multidimensional scaling on the data.

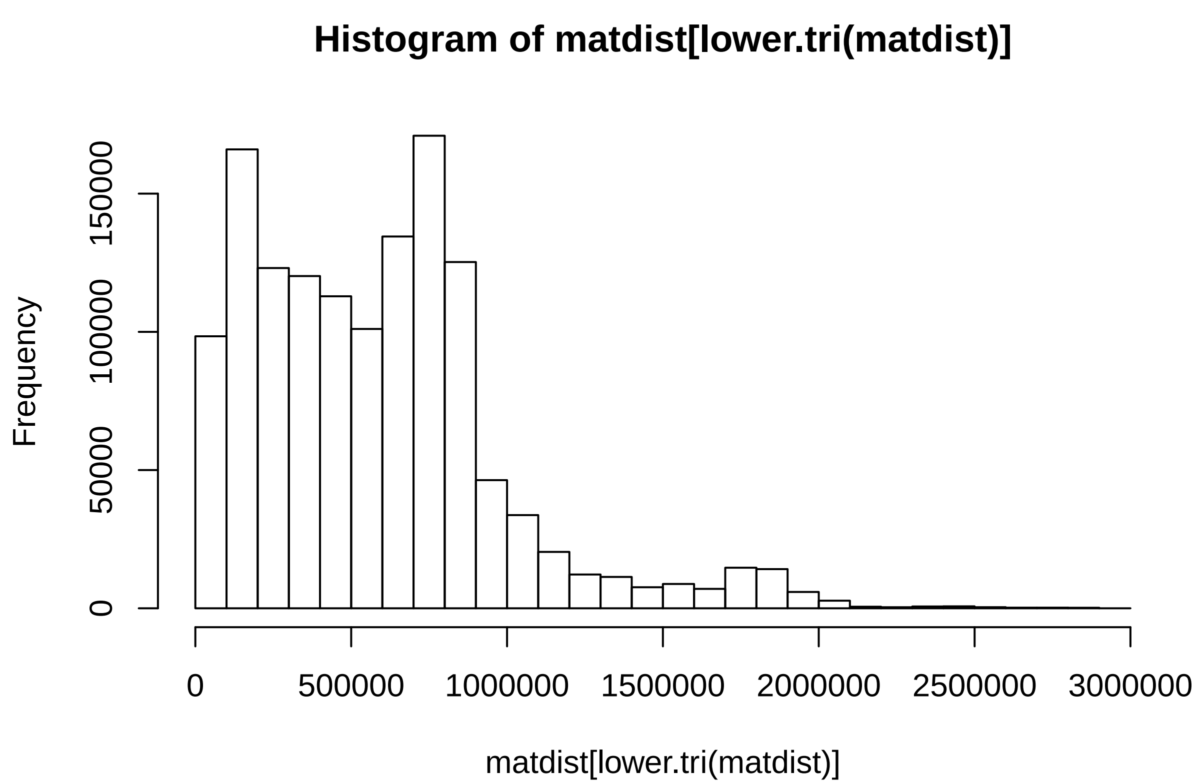
Since computing the MDS for the whole dataset takes too long in a regular computer, we draw an iid sample with 20% of the data.

```
set.seed(12345)

sample <- Xclean[sample(nrow(Xclean),size=as.integer(0.2*nrow(X)),replace=FALSE),]
distances <- dist(sample, method = "euclidean")
matdist <- as.matrix(distances)
matdist[1:5,1:5]
```

```
##             6286        51        720     730      5340
## 6286        0.00  673942.4   73621.99 1112178  486866.4
## 51     673942.43       0.0  707557.91 1774895  221367.0
## 720     73621.99  707557.9       0.00 1070381  529567.6
## 730   1112177.78 1774894.8 1070380.73       0 1597771.3
## 5340   486866.37  221367.0  529567.64 1597771       0.0
```

```
hist(matdist[lower.tri(matdist)])
```

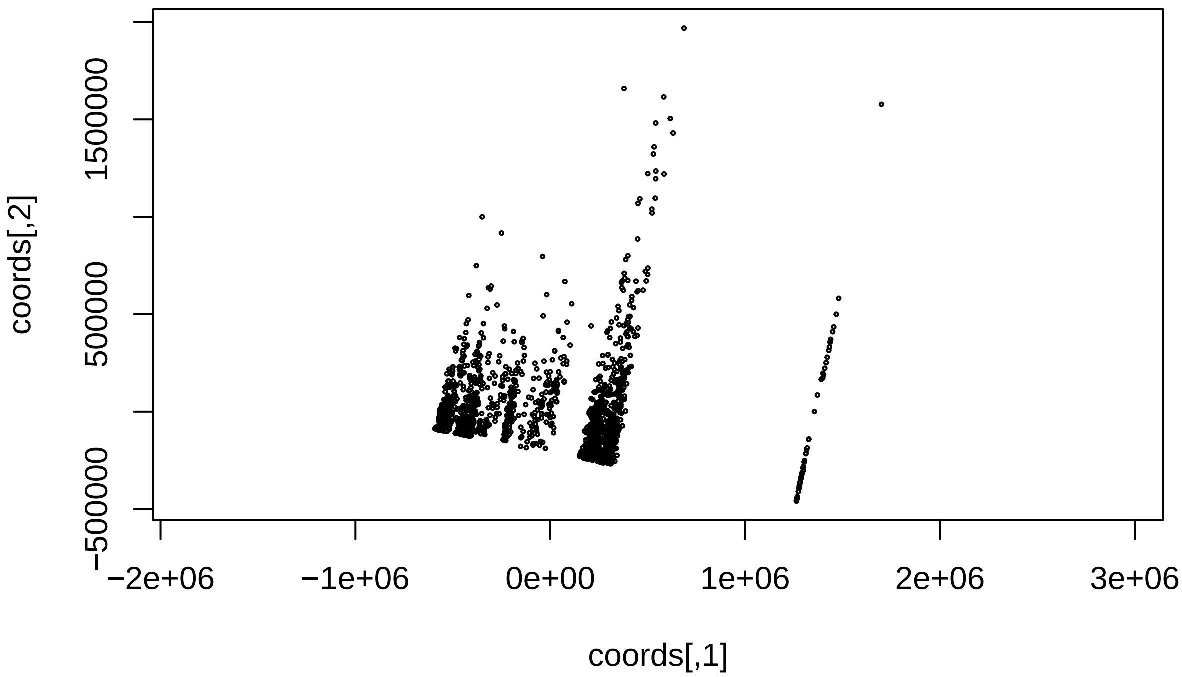


Plot the coordinates in 2D space.

```
mds.out <- cmdscale(matdist, eig=TRUE, k=2)

coords <- mds.out$points

plot(coords, asp=1, cex=0.25)
```



Indeed, there seem two be three differentiated groups.

Let's see if we can accurately differentiate between "active" and "non-active" CPUs

We will try out several classification models:

- Linear Discriminant Analysis (LDA)
- Quadratic Discriminant Analysis (QDA)
- kNN
- Logistic regression
- MLP neural netwrok

```
Xcl <- Xtrain
Xcl$active <- 0
Xcl[Xcl[,"usr"] > 2,]$active <- 1
Xcl <- Xcl[,-16]

Xcl_test <- Xtest
Xcl_test$active <- 0
Xcl_test[Xcl_test[,"usr"] > 2,]$active <- 1
Xcl_test <- Xcl_test[,-16]
```

```
cor(Xclean[,c(1:16)])[16,]
```

```
##       lread      lwrite       scall      swrite        exec       rchar       wchar
```

```
## -0.1413939 -0.1112134 -0.3231884 -0.2722518 -0.2885262 -0.3291151 -0.2890496
##      pgout     pgfree       atch       pgin       vflt     runqsz    freemem
## -0.2218768 -0.2162781 -0.1250742 -0.2417196 -0.4206853 -0.6296624  0.2703083
##   freeswap        usr
##  0.6785262  1.0000000
```

To classify, we use only the variables runqsz and freeswap, which have the highest correlation with our target.

Scaled data for the LDA. By scaling the data, we ensure that both populations have the same variance and covariance, which is a condition for LDA.

```
Xcl_sc <- as.data.frame(scale(Xtrain))
Xcl_sc$usr <- Xtrain$usr
Xcl_sc$active <- 0
Xcl_sc[Xcl_sc[,"usr"] > 2,]$active <- 1
Xcl_sc <- Xcl_sc[,-16]
```

kNN classifier function:

```
# This function returns the error obtained with a kNN classifier with a certain
# number of neighbours (myneighbours), for a certain data set (mydata, mytargets).
loop.k <- function (mydata, mytargets, myneighbours)
{
  errors <- matrix (nrow=length(myneighbours), ncol=2)
  colnames(errors) <- c("k","LOOCV error")

  for (k in myneighbours)
  {
    myknn.cv <- knn.cv (mydata, mytargets, k = myneighbours[k])

    # fill in number of neighbours and LOOCV error
    errors[k, "k"] <- myneighbours[k]

    tab <- table(Truth=mytargets, Preds=myknn.cv)
    (errors[k, "LOOCV error"] <- 1 - sum(tab[row(tab)==col(tab)])/sum(tab))
  }
  errors
}
```

Let's find the best k for kNN using only variables runqsz and freeswap:

```
N <- nrow(Xcl)
neighbours <- 1:10

#try without scaling data
print(loop.k(Xcl[,c(13,15)], Xcl$active, neighbours))
```

```
##        k LOOCV error
##  [1,]  1           0
##  [2,]  2           0
##  [3,]  3           0
##  [4,]  4           0
##  [5,]  5           0
##  [6,]  6           0
##  [7,]  7           0
##  [8,]  8           0
##  [9,]  9           0
```

```
## [10,] 10           0
```

```r
#try scaling data
print(loop.k(scale(Xcl[,c(13,15)]), Xcl$active, neighbours))
```

```
##        k LOOCV error
##  [1,]  1           0
##  [2,]  2           0
##  [3,]  3           0
##  [4,]  4           0
##  [5,]  5           0
##  [6,]  6           0
##  [7,]  7           0
##  [8,]  8           0
##  [9,]  9           0
## [10,] 10           0
```

It looks like kNN is able to classify for any k.

Are variables independent, given their class? (This is the assumption behind the naïve Bayes):

```r
mosthighlycorrelated(Xcl[Xcl[,"active"] == 1,], 20)
```

```
##     First.Variable Second.Variable Correlation
## 136          pgout          pgfree   0.7211286
## 181           exec            vflt   0.6990867
## 51           scall          swrite   0.6371906
## 238        freemem        freeswap   0.6250325
## 17           lread          lwrite   0.6034440
## 179          scall            vflt   0.5333418
## 169         pgfree            pgin   0.5262004
## 102          rchar           wchar   0.4611457
## 227          scall        freeswap  -0.4481595
## 180         swrite            vflt   0.4450407
## 100         swrite           wchar   0.4341828
## 211          scall         freemem  -0.3914750
## 168          pgout            pgin   0.3857962
## 83           scall           rchar   0.3615570
## 182          rchar            vflt   0.3565621
## 84          swrite           rchar   0.3546624
## 195          scall          runqsz   0.3451092
## 67           scall            exec   0.3200894
## 235           pgin        freeswap  -0.3198855
## 185         pgfree            vflt   0.3076171
```

```r
mosthighlycorrelated(Xcl[Xcl[,"active"] == 0,], 20)
```

```
##     First.Variable Second.Variable Correlation
## 238        freemem        freeswap  -0.9972760
## 102          rchar           wchar   0.9185171
## 17           lread          lwrite   0.8662381
## 220           vflt         freemem  -0.8403532
## 236           vflt        freeswap   0.8373249
## 136          pgout          pgfree   0.7728942
## 228         swrite        freeswap   0.7692014
## 212         swrite         freemem  -0.7610032
## 51           scall          swrite   0.7597478
```

```
## 227          scall       freeswap   0.7530421
## 211          scall        freemem  -0.7501177
## 180         swrite           vflt   0.6312700
## 179          scall           vflt   0.5948752
## 169         pgfree           pgin   0.4704214
## 181           exec           vflt   0.4674679
## 219           pgin        freemem  -0.3626794
## 200          pgout         runqsz  -0.3585088
## 235           pgin       freeswap   0.3560449
## 213           exec        freemem  -0.3429921
## 209          lread        freemem  -0.3404348
```

No, they are not independent, therefore it does not make sense to run a Naïve Bayes classifier.

Prepare data for the MLP classifier with a single hidden layer

```
Xcl_nnet <- cbind(Xcl[,c(13,15)], as.factor(as.character(Xcl[,16])))
colnames(Xcl_nnet)[3] <- "active"
```

```
Xcl_nnet_sc <- Xcl_nnet
Xcl_nnet_sc[,c(1,2)] <- scale(Xcl_nnet_sc[,c(1,2)])
```

Cross-Validation of LDA, QDA, kNN, Logistic Regression and MLP

Candidates:

- LDA with runqsz and freeswap as explanatory variables and scaled data
- QDA with runqsz and freeswap as explanatory variables
- kNN with runqsz and freeswap as explanatory variables and one neighbour
- Logistic regression with runqsz and freeswap as explanatory variables
- MLPNN with 5 hidden neurons.

```
set.seed(12345)
k <- 10
CV.folds <- generateCVRuns (Xcl$active, ntimes=1, nfold=k, stratified=TRUE)

cv.results <- matrix (rep(0,7*k),nrow=k)
colnames (cv.results) <- c("k","fold","CV error|LDA","CV error|QDA","CV error|kNN",
                           "CV error|logist", "CV error|nnet")

cv.results[,"k"] <- k

for (j in 1:k)
{
  # get TE data
  te <- unlist(CV.folds[[1]][[j]])

  #Data for nnet
  Xcl_nnet_sc_tr <- Xcl_nnet[-te,]
  Xcl_nnet_sc_te <- Xcl_nnet[te,]

  # train on TR data
  my_lda <- lda(active~. , data=Xcl_sc[-te,c(13,15,16)])
  my_qda <- qda(active~. , data=Xcl[-te, c(13,15,16)])
  my_logist <- glm(active~. , data=Xcl[-te, c(13,15,16)], family=binomial)
```

```r
  my_nnet  <- nnet(active ~. , data = Xcl_nnet_sc_tr, size = 5, trace=F)


  # predict on TE data
  pred_lda <- predict(my_lda, Xcl_sc[te, c(13,15)])
  pred_qda <- predict(my_qda, Xcl[te, c(13,15)])
  pred_knn <- knn(Xcl[-te, c(13,15)], Xcl[te, c(13,15)], cl=Xcl[-te,]$active, k=1)
  pred_logist <- predict(my_logist, Xcl[te, c(13,15)], ty="response")
  pred_nnet  <- round(predict(my_nnet, Xcl_nnet_sc_te[,c(1,2)]))

  # record validation error for this fold
  ct_lda <- table(Truth=Xcl_sc[te,]$active, Pred=pred_lda$class)
  cv.results[j,"CV error|LDA"] <- 1-sum(diag(ct_lda))/sum(ct_lda)

  ct_qda <- table(Truth=Xcl[te,]$active, Pred=pred_qda$class)
  cv.results[j,"CV error|QDA"] <- (1-sum(diag(ct_qda))/sum(ct_qda))

  ct_knn <- table(Truth=Xcl[te,]$active, Pred=pred_knn)
  cv.results[j,"CV error|kNN"] <- 1-sum(diag(ct_knn))/sum(ct_knn)

  ct_logist <- table(truth=Xcl[te,]$active, Pred=round(pred_logist))
  cv.results[j,"CV error|logist"] <- 1-sum(diag(ct_logist))/sum(ct_logist)

  ct_nnet <- table(truth=as.numeric(as.character(Xcl_nnet_sc_te$active)), Pred=pred_nnet)
  cv.results[j, "CV error|nnet"] <- 1-sum(diag(ct_nnet))/sum(ct_nnet)


  cv.results[j,"fold"] <- j
}
(colMeans(cv.results[,c("CV error|LDA", "CV error|QDA", "CV error|kNN",
                        "CV error|logist", "CV error|nnet")]))
```

```
##    CV error|LDA    CV error|QDA    CV error|kNN CV error|logist   CV error|nnet
##     0.006226102     0.000000000     0.000000000     0.000000000     0.000000000
```

This is an easy classification problem, and several models obtain 0 CV error.

Computation of Testing Error for the Chosen Classifiers

QDA

```r
my_qda <- qda(active~. , data=Xcl[, c(13,15,16)])
pred_qda <- predict(my_qda, Xcl_test[, c(13,15,16)], ty="response")

(ct_qda <- table(Truth=Xcl_test$active, Pred=pred_qda$class))
```

```
##      Pred
## Truth    0    1
##     0  109    0
##     1    0 2622
```

```r
(Mp <- 1-sum(diag(ct_qda))/sum(ct_qda))
```

```
## [1] 0
```

kNN

```r
pred_knn <- knn(Xcl[, c(13,15)], Xcl_test[, c(13,15)], cl=Xcl$active, k=1)


ct_knn <- table(Truth=Xcl_test$active, Pred=pred_knn)
(Mp <- 1-sum(diag(ct_knn))/sum(ct_knn))
```

```
## [1] 0
```

Logistic regression

```r
my_logist <- glm(active~ ., data=Xcl[, c(13,15,16)], family=binomial)
pred_logist <- predict(my_logist, Xcl_test[, c(13,15,16)], ty="response")

(ct_logist <- table(truth=Xcl_test$active, pred=round(pred_logist)))
```

```
##      pred
## truth    0    1
##     0  109    0
##     1    0 2622
```

```r
(Mp <- 1-sum(diag(ct_logist))/sum(ct_logist))
```

```
## [1] 0
```

Since we classify cpu's as "active" or "inactive", we have to adjust a regression only on the active ones, and assume that inactive cpu's have a 'usr' equal to 0.

```r
Xcpu <- Xtrain[Xtrain[,"usr"]>2,]
Xno_cpu <- Xtrain[Xtrain[,"usr"]<=2,]
```

Let's adjust some regression models

We can obtain a reduced formula, that is, select some of the features, with the `step()` function.

```r
# Apply step() on binomial(logit)
mod_logit <- glm(usr/100~. , family=binomial(link=logit), data=Xcpu)

suppressWarnings(form_logit <- step(mod_logit, trace=FALSE)$formula)
```

```r
# Apply step() on binomial(probit)
mod_probit <- glm(usr/100~. , family=binomial(link=probit), data=Xcpu)

suppressWarnings(form_probit <- step(mod_probit, trace=FALSE)$formula)
```

```r
# Apply step() on binomial(cloglog)
mod_cloglog <- glm(usr/100~. , family=binomial(link=cloglog), data=Xcpu)

suppressWarnings(form_cloglog <- step(mod_cloglog, trace=FALSE)$formula)
```

Cross-Validation to Choose the Binomials's Link Function

```r
set.seed(12345)
k <- 10
CV.folds <- generateCVRuns (Xcpu$usr, ntimes=1, nfold=k, stratified=TRUE)
```

```r
cv.results <- matrix (rep(0,5*k),nrow=k)
colnames (cv.results) <- c("k","fold", "CV error|logit",
                          "CV error|probit", "CV error|cloglog")
cv.results[,"CV error|logit"] <- 0
cv.results[,"CV error|probit"] <- 0
cv.results[,"CV error|cloglog"] <- 0
cv.results[,"k"] <- k


for (j in 1:k)
{
  # get TE data
  te <- unlist(CV.folds[[1]][[j]])

  # train on TR data
  mod_logit <- glm(form_logit , family=binomial(link=logit), data=Xcpu[-te,])
  mod_probit <- glm(form_probit , family=binomial(link=probit), data=Xcpu[-te,])
  mod_cloglog <- glm(form_cloglog , family=binomial(link=cloglog), data=Xcpu[-te,])


  # predict TE data
  pred_logit <- predict(mod_logit, newdata=Xcpu[te,-16], ty="response")
  pred_probit <- predict(mod_probit, newdata=Xcpu[te,-16], ty="response")
  pred_cloglog <- predict(mod_cloglog, newdata=Xcpu[te,-16], ty="response")


  # record validation error for this fold
  n <- nrow(Xcpu[te,])
  cv.results[j,"CV error|logit"] <- sum((Xcpu[te,]$usr-pred_logit*100)^2) / n
  cv.results[j,"CV error|probit"] <- sum((Xcpu[te,]$usr-pred_probit*100)^2) / n
  cv.results[j,"CV error|cloglog"] <- sum((Xcpu[te,]$usr-pred_cloglog*100)^2) / n

  cv.results[j,"fold"] <- j
}
colMeans(cv.results[, 3:5])
```

```
##   CV error|logit  CV error|probit CV error|cloglog
##        12.402639        11.222247         9.843508
```

Choose cloglog as the link function for the binomial regression

Linear model:

```r
# Apply step() on lm
mod <- glm(usr/100~. , data=Xcpu)

suppressWarnings(form <- step(mod, trace=FALSE)$formula)
```

Regression without classifier:

```r
# lm (Gaussian family and identity link)
mod_noCl <- glm(usr/100~. , data=Xtrain)
suppressWarnings(form_noCL <- step(mod_noCl, trace=FALSE)$formula)

# glm (Binomial family and cloglog link)
```

```
mod_cloglog_noCl <- glm(usr/100~. , family=binomial(link=cloglog), data=Xtrain)
suppressWarnings(form_cloglog_noCl <- step(mod_cloglog_noCl, trace=FALSE)$formula)
```

Ridge regression with classifier:

```
set.seed(12345)
# reference: https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net
#X will be standardized in the modelling function
# Setting alpha = 0 implements ridge regression
ridge_cv <- cv.glmnet(as.matrix(Xcpu[,-16]), Xcpu[,16], alpha=0,
                      standardize=TRUE, nfolds=10)

#get best lambda
(lambda_cv <- ridge_cv$lambda.min)
```

```
## [1] 0.7844842
```

```
#We are going to scale training and testing data separately,
# because we don't want training data to influence testing data
# (recall that when scaling we substract the mean and divide by the std).
```

LASSO regression:

```
Xfull <- X
Xfull_tr <- Xfull[train,]
Xfull_tr_cpu <- Xfull_tr[Xfull_tr$usr > 2,]

# Setting alpha = 1 implements LASSO regression
lasso_cv <- cv.glmnet(as.matrix(Xfull_tr_cpu[,-22]), Xfull_tr_cpu[,22], alpha=1,
                      standardize=TRUE, nfolds=10)

#get best mu
(mu_cv <- lasso_cv$lambda.min)
```

```
## [1] 0.01854905
```

RBF Neural Network:

```
M <- floor(nrow(Xcpu)^(1/3)) # Number of centroids for the RBFNN
```

Cross-Validation of:

- with classifier: Binomial, Normal, Ridge, Null model (predicts the average), MLPNN, LASSO and RBFNN
- withot classifier: Normal and Binomial

```
set.seed(12345)
k <- 10
CV.folds <- generateCVRuns(Xtrain$usr, ntimes=1, nfold=k)

cv.results <- matrix (rep(0,11*k),nrow=k)
colnames (cv.results) <- c("k","fold", "CV error|Bin", "CV error|Normal",
                           "CV error|NormalnoCl", "CV error|BinnoCl",
                           "CV error|Ridge", "CV error|Nul", "CV error|LASSO",
                           "CV error|nnet", "CV error|rbf")
cv.results[,"CV error|Bin"] <- 0
cv.results[,"CV error|NormalnoCl"] <- 0
```

```
cv.results[,"CV error|BinnoCl"] <- 0
cv.results[,"CV error|Ridge"] <- 0
cv.results[,"CV error|Nul"] <- 0
cv.results[,"CV error|LASSO"] <- 0
cv.results[,"CV error|nnet"] <- 0
cv.results[,"CV error|rbf"] <- 0
cv.results[,"k"] <- k


for (j in 1:k)
{
  # get TE data
  te <- unlist(CV.folds[[1]][[j]])

  Xtr <- Xtrain[-te,]
  Xtr_cpu <- Xtr[Xtr$usr > 2,]
  Xtr_no_cpu <- Xtr[Xtr$usr <= 2,]

  Xte <- Xtrain[te,]
  Xte_cpu <- Xte[Xte$usr > 2,]
  Xte_no_cpu <- Xte[Xte$usr <= 2,]

  # Data for LASSO
  Xf_tr <- Xfull_tr[-te,]
  Xf_tr_cpu <- Xf_tr[Xf_tr$usr > 2,]
  Xf_tr_no_cpu <- Xf_tr[Xf_tr$usr <= 2,]

  Xf_te <- Xfull_tr[te,]
  Xf_te_cpu <- Xf_te[Xf_te$usr > 2,]
  Xf_te_no_cpu <- Xf_te[Xf_te$usr <= 2,]

  # Data for MLPNN
  Xtr_cpu_sc <- Xtr_cpu
  Xtr_cpu_sc[,-16] <- scale(Xtr_cpu_sc[,-16])

  Xte_cpu_sc <- Xte_cpu
  Xte_cpu_sc[,-16] <- scale(Xte_cpu_sc[,-16])


  # train on TR data
  mod_cloglog <- glm(form_cloglog, family=binomial(link=cloglog), data=Xtr_cpu)
  mod <- glm(form , data=Xtr_cpu)
  mod_noCl <- glm(form_noCL ,data=Xtr)
  mod_cloglog_noCl <- glm(form_cloglog_noCl, family=binomial(link=cloglog), data=Xtr)
  ridge <- glmnet(as.matrix(Xtr_cpu[,-16]), Xtr_cpu$usr, alpha = 0,
                  lambda = lambda_cv, standardize = TRUE)
  m0 <- lm(usr ~ 1, data = Xtr_cpu)
  lasso <- glmnet(as.matrix(Xf_tr_cpu[,-22]), Xf_tr_cpu$usr, alpha = 1,
                  lambda = mu_cv, standardize = TRUE)
  my_nnet  <- nnet(usr ~. , data = Xtr_cpu_sc, size = 13, trace=F, linout=TRUE,
                   maxit = 500, skip=TRUE)
  my_rbf <- rbf(Xtr_cpu_sc[,-16], Xtr_cpu_sc[,16], size=c(M), maxit=100,
                initFunc="RBF_Weights", linOut=TRUE)
```

```r
  # predict TE data
  pred_cloglog <- predict(mod_cloglog, newdata=Xte_cpu[,-16], ty="response")
  pred <- predict(mod, newdata=Xte_cpu[,-16], ty="response")
  pred_noCl <- predict(mod, newdata=Xte[,-16], ty="response")
  pred_cloglog_noCl <- predict(mod_cloglog_noCl, newdata=Xte[,-16], ty="response")
  y_ridge <- predict(ridge, as.matrix(Xte_cpu[,-16]), ty="response")
  mean <- m0$coefficients
  y_lasso <- predict(lasso, as.matrix(Xf_te_cpu[,-22]), ty="response")
  pred_nnet  <- predict(my_nnet, Xte_cpu_sc)
  pred_rbf  <- predict(my_rbf, Xte_cpu[,-16])

  # record validation error for this fold
  cv.results[j,"CV error|Bin"] <- (sum((Xte_cpu$usr-pred_cloglog*100)^2)
                                   + sum((Xte_no_cpu$usr-0)^2)) / nrow(Xte)
  cv.results[j,"CV error|Normal"] <- (sum((Xte_cpu$usr-pred*100)^2)
                                      + sum((Xte_no_cpu$usr-0)^2)) / nrow(Xte)
  cv.results[j,"CV error|NormalnoCl"] <- sum((Xte$usr-pred_noCl*100)^2) / nrow(Xte)
  cv.results[j,"CV error|BinnoCl"] <- sum((Xte$usr-pred_cloglog_noCl*100)^2) / nrow(Xte)
  cv.results[j,"CV error|Ridge"] <- (t(Xte_cpu$usr - y_ridge) %*% (Xte_cpu$usr - y_ridge)
                                     + sum((Xte_no_cpu$usr-0)^2)) / nrow(Xte)
  cv.results[j,"CV error|Nul"] <- (sum((Xte_cpu$usr-mean)^2)
                                   + sum((Xte_no_cpu$usr-0)^2)) / nrow(Xte)
  cv.results[j,"CV error|LASSO"] <- ((t(Xf_te_cpu$usr - y_lasso) %*% (Xf_te_cpu$usr - y_lasso))
                                     + sum((Xf_te_no_cpu$usr-0)^2)) / nrow(Xte)
  cv.results[j,"CV error|nnet"] <- (sum((Xte_cpu_sc$usr-pred_nnet)^2)
                                    + sum((Xte_no_cpu$usr-0)^2)) / nrow(Xte)
  cv.results[j,"CV error|rbf"] <- (sum((Xte_cpu[,16] - pred_rbf)^2)
                                   + sum((Xte_no_cpu$usr-0)^2))/(nrow(Xte_cpu))

  cv.results[j,"fold"] <- j
}
colMeans(cv.results[, 3:11])
```

```
##        CV error|Bin     CV error|Normal CV error|NormalnoCl     CV error|BinnoCl
##            9.520326            7.162341          266.697770            10.676292
##      CV error|Ridge         CV error|Nul       CV error|LASSO        CV error|nnet
##            7.421414           77.012760            6.664998            28.991335
##        CV error|rbf
##          279.991630
```

The model with the lowest CV error is the LASSO, followed by the ridge regression. It seems that regularization pays off in this scenario.

Compute Testing Error and Confidence Interval

The final model is the LASSO. Let's train it on the whole training dataset and compute its testing error.

Compute Testing Error

```r
Xfull <- X

Xfull_tr <- Xfull[train,]
Xfull_tr_cpu <- Xfull_tr[Xfull_tr$usr > 2,]
```

```r
Xfull_te <- Xfull[-train,]
Xfull_te_cpu <- Xfull_te[Xfull_te$usr > 2,]
Xfull_te_no_cpu <- Xfull_te[Xfull_te$usr <= 2,]

# Setting alpha = 1 implements LASSO regression
my_lasso <- glmnet(as.matrix(Xfull_tr_cpu[,-22]), Xfull_tr_cpu$usr, alpha = 1,
                   lambda = mu_cv, standardize = TRUE)
y_lasso <- predict(my_lasso, as.matrix(Xfull_te_cpu[,-22]), ty="response")

Mp <- ((t(Xfull_te_cpu$usr-y_lasso) %*% (Xfull_te_cpu$usr-y_lasso))
       + sum((Xfull_te_no_cpu$usr-0)^2))

N <- nrow(Xfull_te)
(NRMSE <- sqrt(Mp / ((N-1)*var(Xfull_te$usr))))
```

```
##            s0
## s0 0.1350015
```

```r
(R2 <- 1-NRMSE^2) # R^2
```

```
##            s0
## s0 0.9817746
```

We have obtained an accuracy of 98.18%, 2% higher than when using PCA without a previous classifier!

Confidence Interval for the Determination Coefficient R^2

Source: https://stats.stackexchange.com/questions/175026/formula-for-95-confidence-interval-for-r2
Signification level: 5%

```r
k <- sum(my_lasso$beta > 1e-9 | my_lasso$beta < -1e-9) #number of predictors of our model

SE <- sqrt( (4*R2*(1-R2)^2*(N-k-1)^2) / ((N^2-1)*(3+N)) )


c(R2-2*SE, R2+2*SE)
```

```
## [1] 0.9804017 0.9831475
```

By splitting the initial dataset into two separate groups, we have managed to obtain a 2% increase in accuracy in the final regression, at the cost of higher model complexity.