Modelling of CPU usage through PCA analysis

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 obtain the principal components of the data through PCA, and then use a few of these components to train a simple model. Several models are compared through a 10fold CV procedure.

```
library(TunePareto) # for generateCVRuns()
library(glmnet) # ridge regression
library(RSNNS) # MLP, RBFNN
```

Read data

```
X <- read.csv('cpu.csv')
# source: https://www.openml.org/d/197
```

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.

To get a better grasp of how our data looks like, it is useful to project it into a 2D space with a multidimensional scaling (MDS). 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, which will preserve the statistical properties of the data.

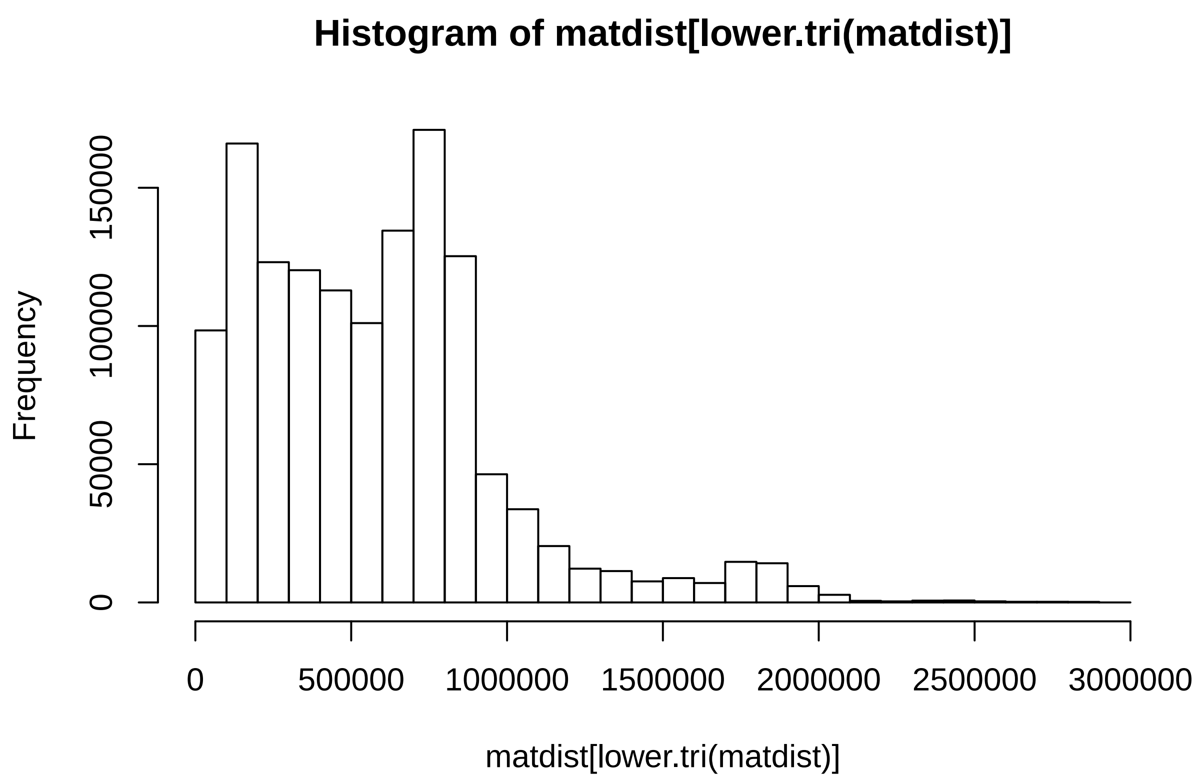
```
set.seed(12345)

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

```r
matdist[1:5,1:5]
```

```
##              6286         51          720      730      5340
## 6286        0.00   673942.5    73622.26 1112178   486866.4
## 51     673942.50        0.0   707557.93 1774895   221367.4
## 720     73622.26   707557.9        0.00 1070381   529567.7
## 730   1112177.79 1774894.8 1070380.73        0 1597771.3
## 5340   486866.38   221367.4   529567.72 1597771        0.0
```

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

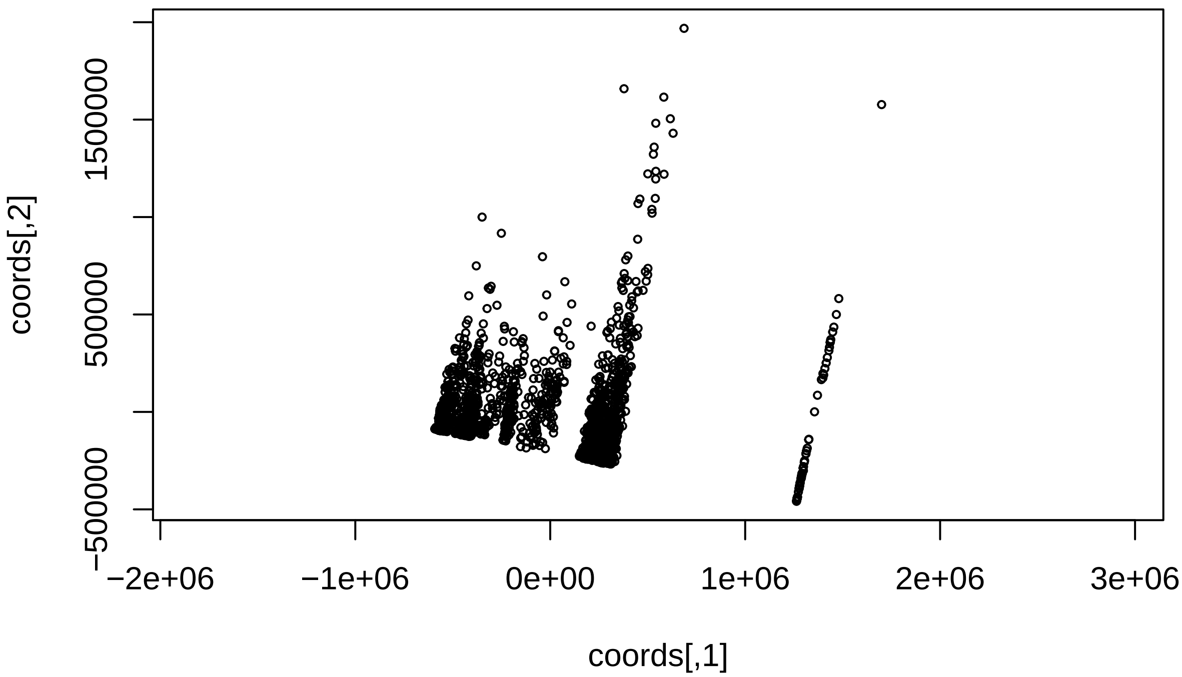


Plot the coordinates in 2D space.

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

  coords <- mds.out$points

  plot(coords, asp=1, cex=0.5)
}
```



We can see that coordinate y behaves very linearly with respect to coordinate x. Moreover, there seem two be three differentiated groups, one much bigger than the other.

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

```
set.seed(12345)

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

Xtrain <- X[train,]
pca_train <- prcomp(Xtrain[,1:21])$x[,1:3]
df_train <- as.data.frame(cbind(pca_train, Xtrain[,22]))


Xtest <- X[-train,]
pca_test <- prcomp(Xtest[,1:21])$x[,1:3]
df_test <- as.data.frame(cbind(pca_test, Xtest[,22]))
```

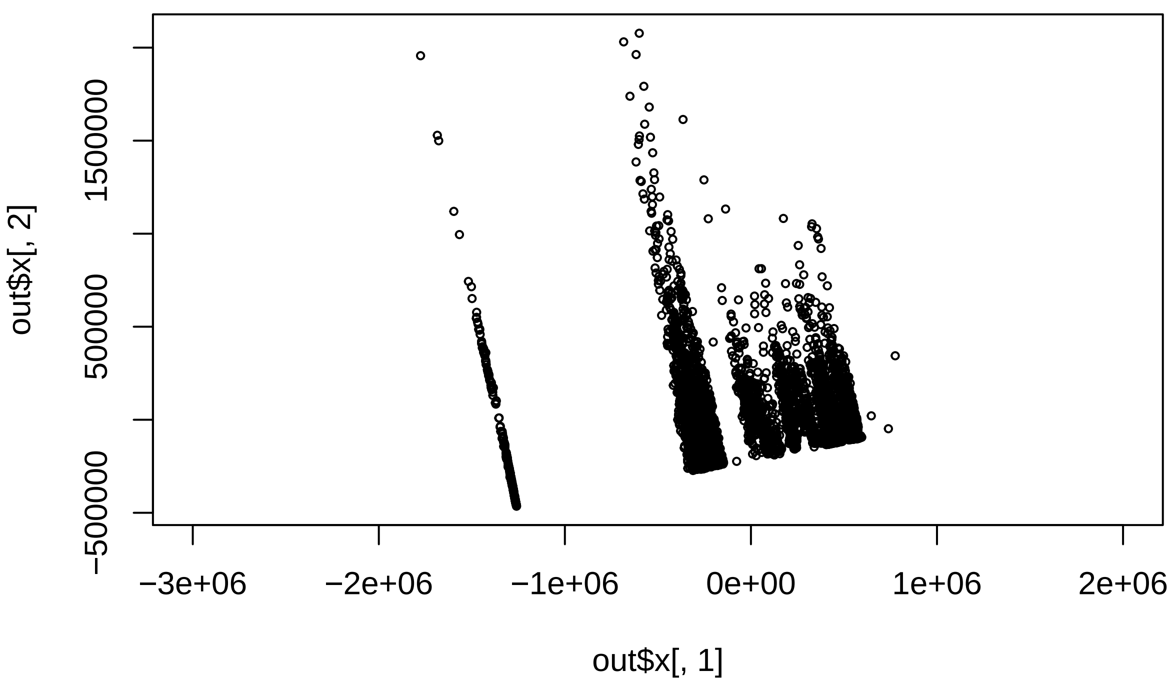
Extract principal components

With PCA, we can visualize the data and also extract features that concentrate most of the variability. It is important to extract principal components not from the whole dataset, but from the training set. This is because PCA is computed taking into account the variability of the data. If we computed PCA with all the data, there would be a data leakage from the testing set to the training set.

```
out <- prcomp(Xtrain[,1:21])  # leave the 22 feature out, as it's the target
cumsum(out$sdev)/sum(out$sdev)
```

```
##  [1] 0.5428061 0.8492298 0.9946281 0.9973049 0.9990717 0.9993137 0.9995003
##  [8] 0.9996337 0.9997283 0.9997963 0.9998606 0.9999020 0.9999295 0.9999549
## [15] 0.9999729 0.9999799 0.9999865 0.9999922 0.9999967 0.9999993 1.0000000
```

```
plot(out$x[,1], out$x[,2], asp=1, cex=0.5)
```



PCA allows to concentrate 99.46% of the variability in the data with just three principal components.

Moreover, the biplot clearly indicates two different groups that are linearly separable. We might be interested in separating them out first and then run two separate regressions, one on each group. But first we should try if a single regression model is good enough.

We now proceed to test various regression models:

- Linear regression with binomial link function
- Linear regression with Gaussian link function
- Ridge regression
- LASSO regression
- MLP neural network
- RBF neural network

Cross-Validation to Choose the Binomial's Link Function

```r
set.seed(12345)
k <- 10
CV.folds <- generateCVRuns (df_train$V4, ntimes=1, nfold=k)

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 validation data
  te <- unlist(CV.folds[[1]][[j]])

  # train on TR data
  mod_logit <- glm(V4/100 ~ ., family=binomial(link=logit), data=df_train[-te,])
  mod_probit <- glm(V4/100 ~ ., family=binomial(link=probit), data=df_train[-te,])
  mod_cloglog <- glm(V4/100 ~ ., family=binomial(link=cloglog), data=df_train[-te,])


  # predict TE data
  pred_logit <- predict(mod_logit, newdata=df_train[te,-4], ty="response")
  pred_probit <- predict(mod_probit, newdata=df_train[te,-4], ty="response")
  pred_cloglog <- predict(mod_cloglog, newdata=df_train[te,-4], ty="response")


  # record validation error for this fold
  n <- nrow(df_train[te,])
  cv.results[j,"CV error|logit"] <- sum((df_train[te,]$V4-pred_logit*100)^2) / n
  cv.results[j,"CV error|probit"] <- sum((df_train[te,]$V4-pred_probit*100)^2) / n
  cv.results[j,"CV error|cloglog"] <- sum((df_train[te,]$V4-pred_cloglog*100)^2) / n

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

```
##   CV error|logit  CV error|probit CV error|cloglog
##         108.7704         122.6367         151.8800
```

The logit link outperforms the rest.

Ridge regression: We need to select a lambda regularization parameter for the model.

```r
set.seed(12345)
# reference: https://www.datacamp.com/community/tutorials/tutorial-ridge-lasso-elastic-net
# Data will be standardized by the modelling function
# Setting alpha = 0 implements ridge regression
ridge_cv <- cv.glmnet(as.matrix(df_train[,-4]), df_train[,4], alpha = 0,
                      standardize = TRUE, nfolds = 10)
```

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

```
## [1] 1.232473
```

```
# We are going be scaling 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: Again, we need to select the mu regularization parameter.

```
set.seed(12345)

# Setting alpha = 1 implements LASSO regression
lasso_cv <- cv.glmnet(as.matrix(df_train[,-4]), df_train[,4], alpha=1,
                      standardize=TRUE, nfolds=10)

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

```
## [1] 0.04640184
```

RBF Neural Network: We select the number of centroids for the RBFNN.

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

Cross-Validation OF Binomial, Gaussian, Ridge, LASSO, MLPNN and RBFNN models

```
set.seed(12345)
k <- 10
CV.folds <- generateCVRuns(df_train$V4, ntimes=1, nfold=k)

cv.results <- matrix (rep(0,8*k),nrow=k)
colnames (cv.results) <- c("k","fold", "CV error|Bin",
                           "CV error|Gaussian", "CV error|Ridge",
                           "CV error|LASSO", "CV error|nnet",
                           "CV error|rbf")
cv.results[,"CV error|Bin"] <- 0
cv.results[,"CV error|Gaussian"] <- 0
cv.results[,"CV error|Ridge"] <- 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 validation data
  te <- unlist(CV.folds[[1]][[j]])

  # train on TR data
  mod_binomial <- glm(V4/100 ~ ., family=binomial(link=logit), data=df_train[-te,])
  mod_Gaussian <- glm(V4/100 ~ ., data=df_train[-te,])
```

```
  ridge <- glmnet(as.matrix(df_train[-te,-4]), df_train[-te,4], alpha=0,
                  lambda=lambda_cv, standardize=TRUE)
  lasso <- glmnet(as.matrix(df_train[-te,-4]), df_train[-te,4], alpha=1,
                  lambda=mu_cv, standardize=TRUE)
  my_nnet <- mlp(df_train[-te,-4], df_train[-te,4], size=c(5, 5), maxit=100,
                 hiddenActFunc="Act_Logistic", linOut=TRUE)
  my_rbf <- rbf(df_train[-te,-4], df_train[-te,4], size=c(M), maxit=100,
                initFunc="RBF_Weights", linOut=TRUE)

  # predict TE data
  pred_binomial <- predict(mod_binomial, newdata=df_train[te,-4], ty="response")
  pred_Gaussian <- predict(mod_Gaussian, newdata=df_train[te,-4])
  y_ridge <- predict(ridge, as.matrix(df_train[te,-4]))
  y_lasso <- predict(lasso, as.matrix(df_train[te,-4]))
  pred_nnet  <- predict(my_nnet, df_train[te,-4])
  pred_rbf  <- predict(my_rbf, df_train[te,-4])

  # record validation error for this fold
  n <- nrow(df_train[te,])
  cv.results[j,"CV error|Bin"] <- sum((df_train[te,4]-pred_binomial*100)^2) / n
  cv.results[j,"CV error|Gaussian"] <- sum((df_train[te,4]-pred_Gaussian*100)^2) / n
  cv.results[j,"CV error|Ridge"] <- (t(df_train[te,4] - y_ridge) %*% (df_train[te,4] - y_ridge)) / n
  cv.results[j,"CV error|LASSO"] <- (t(df_train[te,4] - y_lasso) %*% (df_train[te,4] - y_lasso)) / n
  cv.results[j,"CV error|nnet"] <- sum((df_train[te,4] - pred_nnet)^2) / n
  cv.results[j,"CV error|rbf"] <- sum((df_train[te,4] - pred_rbf)^2) / n

  cv.results[j,"fold"] <- j
}
colMeans(cv.results[, 3:8])
```

```
##      CV error|Bin CV error|Gaussian    CV error|Ridge    CV error|LASSO
##          108.7704          169.8045          170.4616          169.8160
##     CV error|nnet      CV error|rbf
##         1108.7035          325.8977
```

This time it seems that linear models outperform non-linear models! We expected that, as the PCA biplot cleary showed a markedly linear trend. The best model is the glm with logit link.

Compute Testing Error and Confinence Interval

Compute Testing Error

```
final_mod <- glm(V4/100 ~ ., family=binomial(link=logit), data=df_train)
pred <- predict(final_mod, newdata=df_train[,-4], ty="response")

Mp <- sum((df_train[te,4]-pred_binomial*100)^2)

N <- nrow(df_train)
(NRMSE <- sqrt(Mp / ((N-1)*var(df_train[,4]))))
```

```
## [1] 0.17951
```

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

```
## [1] 0.9677761
```

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%

```
n_coeffs <- 4  # number of predictors of our model

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

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

```
## [1] 0.9660623 0.9694900
```

This is a very tight interval, indicating that our model will generalize well.