QSAR model by a LASSO regression

Edward Francisco Mendez-Otalvaro, Daniel Alberto Barragan, Isaias Lans-Vargas

2021

Functions to calculate metrics of the model

The metrics that are going to be used are:

- Root mean square error of training set (RMSEC)
- Root mean square error of test set (RMSEP)
- Square of the correlation coefficient of training set (R2tr).
- Square of the correlation coefficient of test set (Also called Q2F2 coefficient).
- Q2F3 coefficient of Todeschini.
- Spearman correlation coefficient of test set (σ).
- Square of the Spearman correlation coefficient of test set ($\sigma 2$).
- Square of the correlation coefficient of cross validation leaving one outside (R2LOO).
- Root mean square error of cross validation leaving one outside (RMSELOO).

All the above metrics are rotinarious in QSAR modelling, in order to get some criteria about a good predictive or ranking model (our main objetive).

```r
Metricas <- function(obs_tr, pred_tr, obs_out, pred_out) {

    ssr_tr <- sum((obs_tr - pred_tr)^2)
    sst_tr <- sum((obs_tr - mean(obs_tr))^2)

    ssr_out <- sum((obs_out - pred_out)^2)
    sst_out <- sum((obs_out - mean(obs_out))^2)

    n_tr <- 20
    n_out <- 7

    ## RMSEC

    RMSEC <- sqrt(ssr_tr/(n_tr))

    ## RMSEP

    RMSEP <- sqrt(ssr_out/(n_out))

    ## R2tr

    R2tr <- (1 - (ssr_tr/sst_tr))

    ## R2out/Q2F2/R2Tropsha

    R2out <- (1 - (ssr_out/sst_out))
    ## Q2F3

    Q2F3 <- (1 - ((ssr_out/n_out)/(sst_tr/n_tr)))

    ## Sigma

    rho <- cor(obs_out, pred_out, method = c("spearman"))

    ## Sigma2

    rho2 <- (rho)^2

    ## Salida

    output <- list(RMSEC = RMSEC, RMSEP = RMSEP, R2tr = R2tr, R2out = R2out, Q2F3 = Q2F3,
        rho = rho, rho2 = rho2)
    return(output)
}

## LOO Metrics

Metrica_LOO <- function(obs, pred) {

    ssr_LOO <- sum((obs - pred)^2)
    sst_LOO <- sum((obs - mean(obs))^2)
    n_LOO <- 20

    ## R2LOO

    R2LOO <- (1 - (ssr_LOO/sst_LOO))

    ## RMSELOO

    RMSELOO <- sqrt(ssr_LOO/n_LOO)

    output <- list(R2LOO = R2LOO, RMSELOO = RMSELOO)
    return(output)
}
```

Importing dataset

Now, the dataset previously prepared is upload (we concatenate the molecular descriptors with the biological activity):

```
Descriptores_I <- read.csv("Descriptores_Alvrunner.csv", stringsAsFactors = FALSE)

Descriptores_II <- read.csv("Descriptores_MOPAC.csv", stringsAsFactors = FALSE)

Descriptores <- cbind(Descriptores_I, Descriptores_II)


## Biological activity

Actividad <- read.csv2("Actividad.csv", stringsAsFactors = FALSE)
```

```r
## Joining molecular descriptors with activity

Dataset <- cbind(Descriptores, Actividad)

## Cleaning labels

Dataset$ID <- NULL
Dataset$IC50..uM. <- NULL

## Renaming biological activity

names(Dataset)[names(Dataset) == "IC50..M."] <- "pIC50"
```

Converting biological activity (EC50) into logarithmic scale, and then, calculating dimension of the dataframe

```r
Dataset[, 2284] <- -log10(Dataset[, 2284])

dim(Dataset)
```

```
## [1]   27 2284
```

So, there are 27 molecules with 2283 molecular descriptors and a biological activity response.

Pretreatment of the dataset (cleaning)

Let's remove NA columns; columns with variance of zero (constant columns) and columns with more than half filled with zeros (Refer to the Manuscript to the criteria selected).

```r
## Removing NA's
Dataset <- Dataset[, !apply(Dataset, 2, function(x) any(is.na(x)))]

## Removing columns with variance equal to zero.
Dataset <- Dataset[, apply(Dataset, 2, var, na.rm = TRUE) != 0]

## Removing columns with constant values
Dataset <- Dataset[, !apply(Dataset, 2, function(x) length(unique(x)) == 1)]

## Removing columns with more than half filled with zeros
Dataset <- Dataset[, colSums(Dataset != 0) > nrow(Dataset)/2]

dim(Dataset)
```

```
## [1]  27 365
```

Since the dataset is very high dimensional `27 X 365`. Let's calculate a correlation matrix for all the descriptors, and then, let's remove the high correlated molecular descriptors (R2 of Pearson >0.99), with this, the multicolinearity between columns could be improved (a problem that could generate bias in our QSAR model)

```r
## Removing high correlated descriptors
library(caret)

## Correlation matrix calculation
Dataset_Cor = cor(Dataset)

## Removing high correlated descriptors with a R2>0.99
```

```r
hc = findCorrelation(Dataset_Cor, cutoff = 0.99)
hc = sort(hc)

## Non correlated descriptor matrix

Dataset_hc = Dataset[, -c(hc)]

## Dimension of the dataset
dim(Dataset_hc)
```

```
## [1]  27 275
```

So now, the dataset has a dimension of `27 X 275`. The improvement of the descriptors is good, but the high dimensionality still appears (n<p, ie; more descriptors than observations)

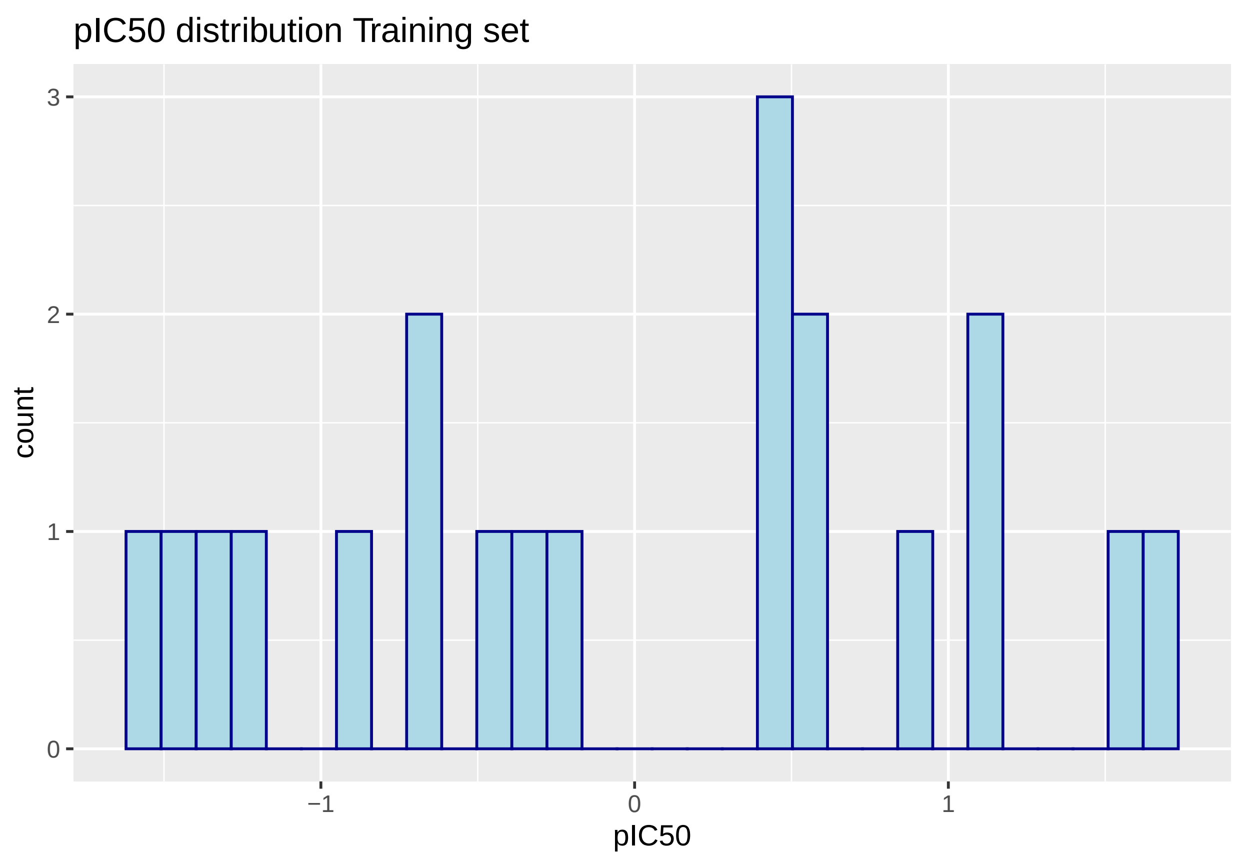
Splitting dataset into training a test set

Since the dataset is very asymmetrical, and there are few observations, let's try to split the dataset in order to get a good balance between both groups (avoiding artifacts by asymetrical splitting). The **caret** function from R allows to carry out this task. Also, the dataset will be scaled subtracting the mean and dividing by standard deviation of data. The ratio will be 70% training and 30% testing.

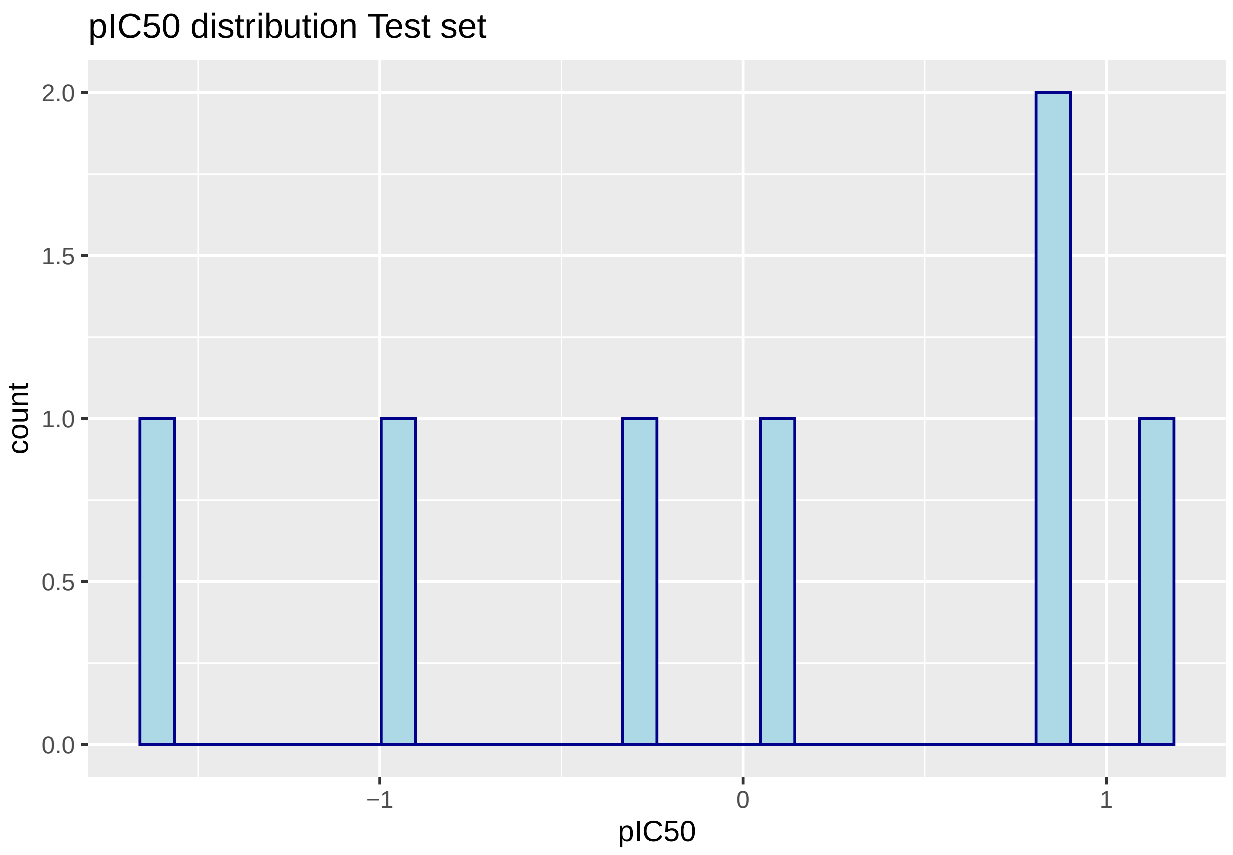
```r
## Scaling
Mean <- apply(Dataset_hc, 2, mean)
SD <- apply(Dataset_hc, 2, sd)

Dataset_S <- as.matrix(scale(Dataset_hc, center = Mean, scale = SD))


centrado <- t(attr(Dataset_S, "scaled:center"))
escalado <- t(attr(Dataset_S, "scaled:scale"))


## Splitting the data in training and test (70% training and 30% test)

set.seed(101)
Muestra <- createDataPartition(Dataset_S[, 275], times = 1, p = 0.7, list = FALSE)

Modelamiento <- as.matrix(Dataset_S[Muestra, ])
Prueba <- as.matrix(Dataset_S[-Muestra, ])
```

Let's apreciate the distribution of the data in the training and test set

```r
library(ggplot2)

## Basic histogram

ggplot(as.data.frame(Modelamiento), aes(x = pIC50)) + geom_histogram(color = "darkblue",
    fill = "lightblue") + ggtitle("pIC50 distribution Training set")
```



```
ggplot(as.data.frame(Prueba), aes(x = pIC50)) + geom_histogram(color = "darkblue",
    fill = "lightblue") + ggtitle("pIC50 distribution Test set")
```



A Student-t statistical test will be carried out to get insight about the differences between groups. The null hypothesis states that the data comes from the same statistical distribution. The alternative hypothesis states that the data does not come from the same statistical distribution. The test will be carried out with a significance level of 95% (p-value = 0.05)

```
t.test(Modelamiento[, 275], Prueba[, 275])
```

```
##
##  Welch Two Sample t-test
##
## data:  Modelamiento[, 275] and Prueba[, 275]
## t = -0.019338, df = 10.475, p-value = 0.9849
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.0032746  0.9859042
## sample estimates:
##    mean of x    mean of y
## -0.002251715  0.006433471
```

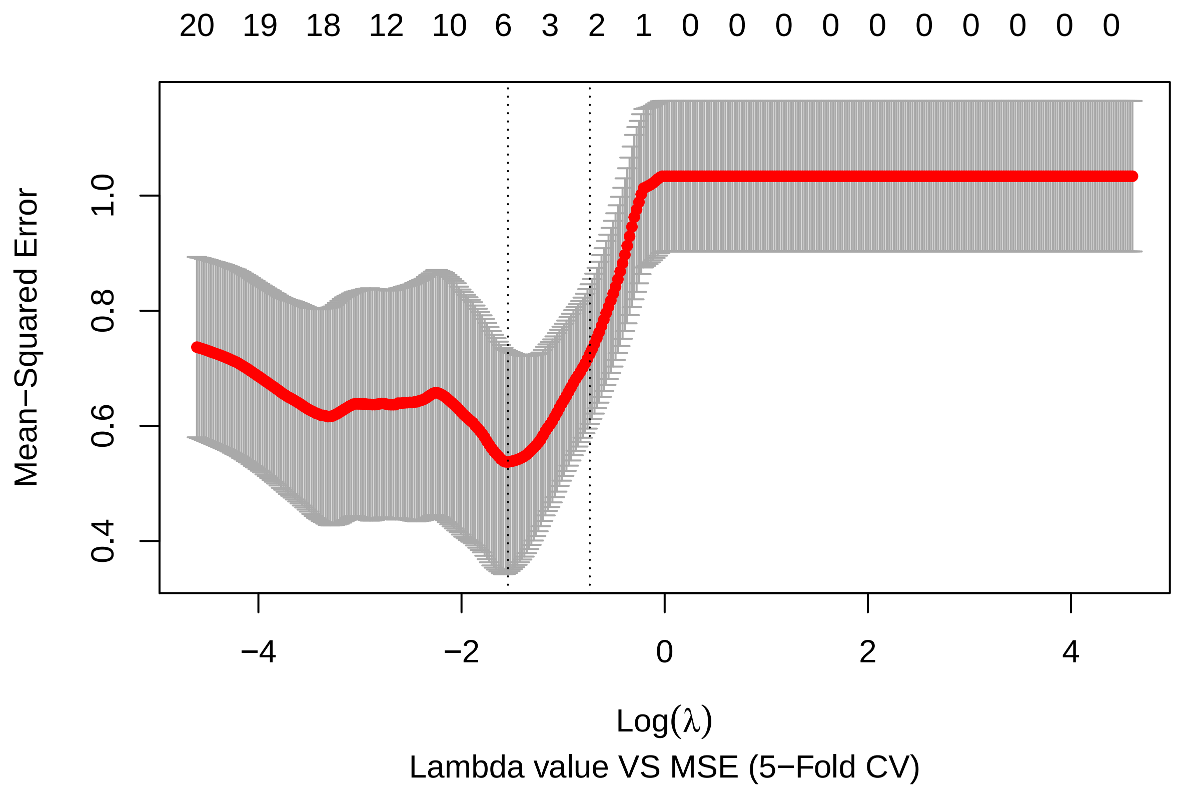
Since `p-value>0.05`, then the null hypothesis is accepted, so both data distributions come from the same statistical distribution with a statistical significance level of 95%.

LASSO Regression

The LASSO regression is a linear penalized regression technique that permits to obtain a linear correlation in a high dimensional system. The algorithm can function both as a feature selection technique and as a linear regression model with the same selected features/descriptors. Please refer to the manuscript to get more info about the algorithm.

Adjusting λ value

```r
## Getting the best lambda parameter for the penalized function by a 5-fold cross
## validation of training set (values of lambda from 0 to 100)

library(glmnet)
set.seed(1e+05)

nFolds <- 5
foldid <- sample(rep(seq(nFolds), length.out = nrow(Modelamiento)))

Modelo_Lasso <- cv.glmnet(Modelamiento[, 1:274], Modelamiento[, 275], alpha = 1,
    nfolds = nFolds, lambda = 10^seq(2, -2, by = -0.01), foldid = foldid, standardize = FALSE)

## Plotting lambda VS mean absolute error (MSE) to get the best lambda to tuning
## the model

plot(Modelo_Lasso, sub = "Lambda value VS MSE (5-Fold CV)")
```



```r
## Best lambda value
Mejor_Lambda <- Modelo_Lasso$lambda.1se
print(Mejor_Lambda)
```

```
## [1] 0.4786301
```

Training the model

```
## Training the model with the best lambda value from the 5-fold cross validation

Mejor_Lasso <- glmnet(Modelamiento[, 1:274], Modelamiento[, 275], alpha = 1, lambda = Mejor_Lambda,
    standardize = FALSE)

## Model info

print(Mejor_Lasso)
```

```
##
## Call:  glmnet(x = Modelamiento[, 1:274], y = Modelamiento[, 275], alpha = 1,      lambda = Mejor_Lam
##
##   Df  %Dev Lambda
## 1  2 46.66 0.4786
```

```
## Features selected by LASSO

Coef_Import = coef(Mejor_Lasso)[, 1]
Coef_Import = sort(abs(Coef_Import), decreasing = T)
Coef_Import <- as.matrix(Coef_Import)
print(Coef_Import[1:5, ])
```

```
##    F04.C.S.    B10.C.S. (Intercept) ALogPS_logP ALogPS_logS
##  0.28912994  0.06287162  0.01078790  0.00000000  0.00000000
```

The LASSO model selected two features from the 274: F04.C.S. & B10.C.S. So the equation looks like:

$$\mathbf{pIC_{50}} = \mathbf{b_0}(\mathbf{F04.C.S.}) + \mathbf{b_1}(\mathbf{B10.C.S}) + \mathbf{b_2}(\mathbf{Intercept}) \quad (1)$$

(2)

Predicting the training dataset with the model (Internal):

```
Predicho_tr <- predict.glmnet(Mejor_Lasso, s = Mejor_Lambda, newx = Modelamiento[,
    1:274])

Observado_tr <- Modelamiento[, 275]
```

Predicting the test dataset with the model (External):

```
Predicho_out <- predict.glmnet(Mejor_Lasso, s = Mejor_Lambda, newx = Prueba[, 1:274])

Observado_out <- Prueba[, 275]
```

Unscaling data

```
## Function to unscaling

Desescalar <- function(X) {
    X_desescalada <- (X * escalado[, 275] + centrado[, 275])
}
```

```r
## Internal prediction

Predicho_tr <- Desescalar(Predicho_tr)
Observado_tr <- Desescalar(Observado_tr)

Obs_df <- as.data.frame(cbind(Observado_tr, Predicho_tr))
colnames(Obs_df) <- c("Observed", "Predicted")
rownames(Obs_df) <- Muestra[1:20]


## External prediction

Predicho_out <- Desescalar(Predicho_out)
Observado_out <- Desescalar(Observado_out)

Pred_df <- as.data.frame(cbind(Observado_out, Predicho_out))
colnames(Pred_df) <- c("Observed", "Predicted")
rownames(Pred_df) <- c("4", "5", "10", "15", "16", "19", "26")
```

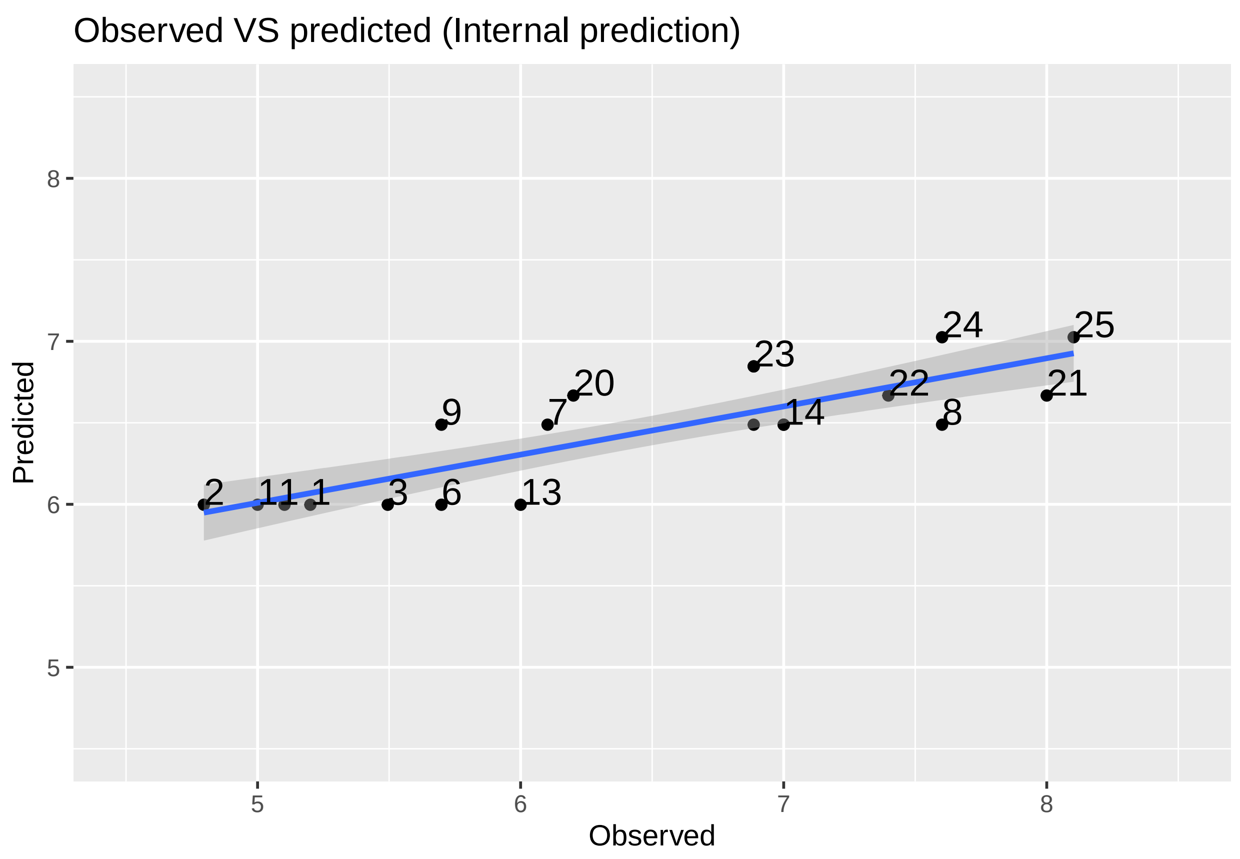
Statistical metrics and plotting of the model

```r
Metricas_Lasso <- Metricas(Observado_tr, Predicho_tr, Observado_out, Predicho_out)

Metricas_Lasso <- (do.call(rbind, Metricas_Lasso))
rownames(Metricas_Lasso) <- c("RMSEC", "RMSEP", "R2tr", "R2out", "Q2F3", "rho", "rho2")
Metricas_Lasso <- round(Metricas_Lasso, 1)
print(Metricas_Lasso)
```

```
##         1
## RMSEC 0.7
## RMSEP 0.8
## R2tr  0.5
## R2out 0.4
## Q2F3  0.4
## rho   0.8
## rho2  0.6
```

```r
library(ggplot2)

# Basic scatter plot Training
ggplot(Obs_df, aes(x = Observed, y = Predicted)) + geom_point() + geom_smooth(method = glm) +
    ggtitle("Observed VS predicted (Internal prediction)") + geom_text(size = 5,
    aes(label = rownames(Obs_df)), hjust = 0, vjust = 0, check_overlap = TRUE) +
    coord_cartesian(xlim = c(4.5, 8.5), ylim = c(4.5, 8.5))
```

```
## `geom_smooth()` using formula 'y ~ x'
```

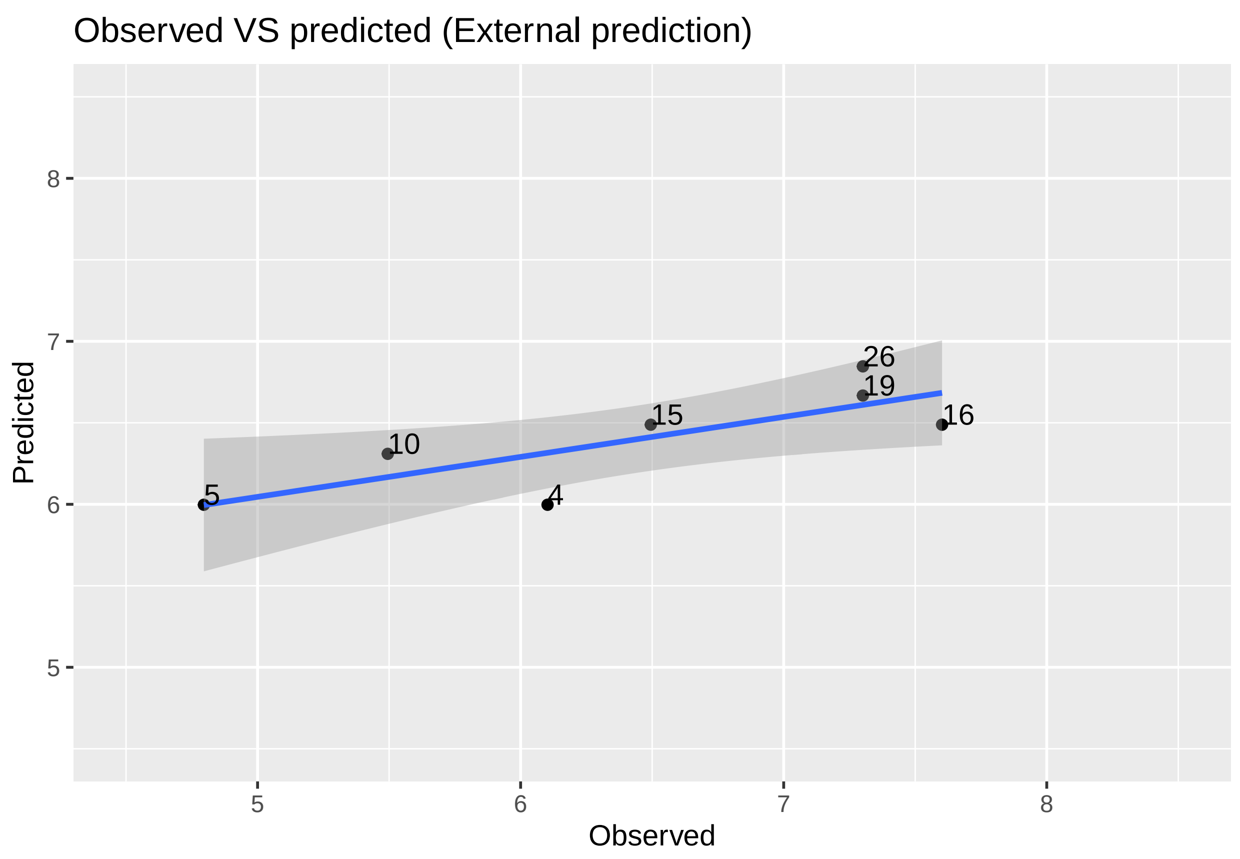


```r
ggsave("lassointernal.png")
```

```
## Saving 6.5 x 4.5 in image
## `geom_smooth()` using formula 'y ~ x'
```

```r
# Basic scatter plot Test
ggplot(Pred_df, aes(x = Observed, y = Predicted)) + geom_point() + geom_smooth(method = lm) +
    ggtitle("Observed VS predicted (External prediction)") + geom_text(aes(label = rownames(Pred_df)),
    check_overlap = TRUE, hjust = 0, vjust = 0) + coord_cartesian(xlim = c(4.5, 8.5),
    ylim = c(4.5, 8.5))
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```r
ggsave("lassoexternal.png")
```

```
## Saving 6.5 x 4.5 in image
## `geom_smooth()` using formula 'y ~ x'
```

Leaving one out cross-validation (LOO-CV) of the data

A statistical metric that is common in QSAR modeling is the RMSE and R2 of the LOO-CV. Let's estimate these parameters

```r
## Storing matrix for the cycle

Pred_CV = matrix()
Modelo_i = list()

## Training set

Modelamiento_CV <- as.matrix(Modelamiento)
set.seed(1e+05)

for (i in 1:nrow(Modelamiento_CV)) {

    ## Removing i row from 1 to the number of rows of the dataset (each cycle left one
    ## out)

    Calibracion_CV <- as.matrix(Modelamiento_CV[-i, ])
    Prueba_CV <- as.matrix(Modelamiento_CV[i, ])
    ## Lambda value getting from the first CV

    Mejor_Lambda_CV <- 0.4786

    ## Training the model

    Modelo_CV <- glmnet(Calibracion_CV[, 1:274], Calibracion_CV[, 275], alpha = 1,
        lambda = Mejor_Lambda_CV, standardize = FALSE)
    ## Predicting the i-row

    Prueba_pred_CV <- t(Prueba_CV[1:274, ])

    Pred_CV_model = predict.glmnet(Modelo_CV, s = Mejor_Lambda_CV, newx = Prueba_pred_CV)

    ## Predicted values for each i-row
    Pred_CV[i] <- Pred_CV_model

    ## Model for each i-row
    Modelo_i[[i]] <- Modelo_CV
}
```

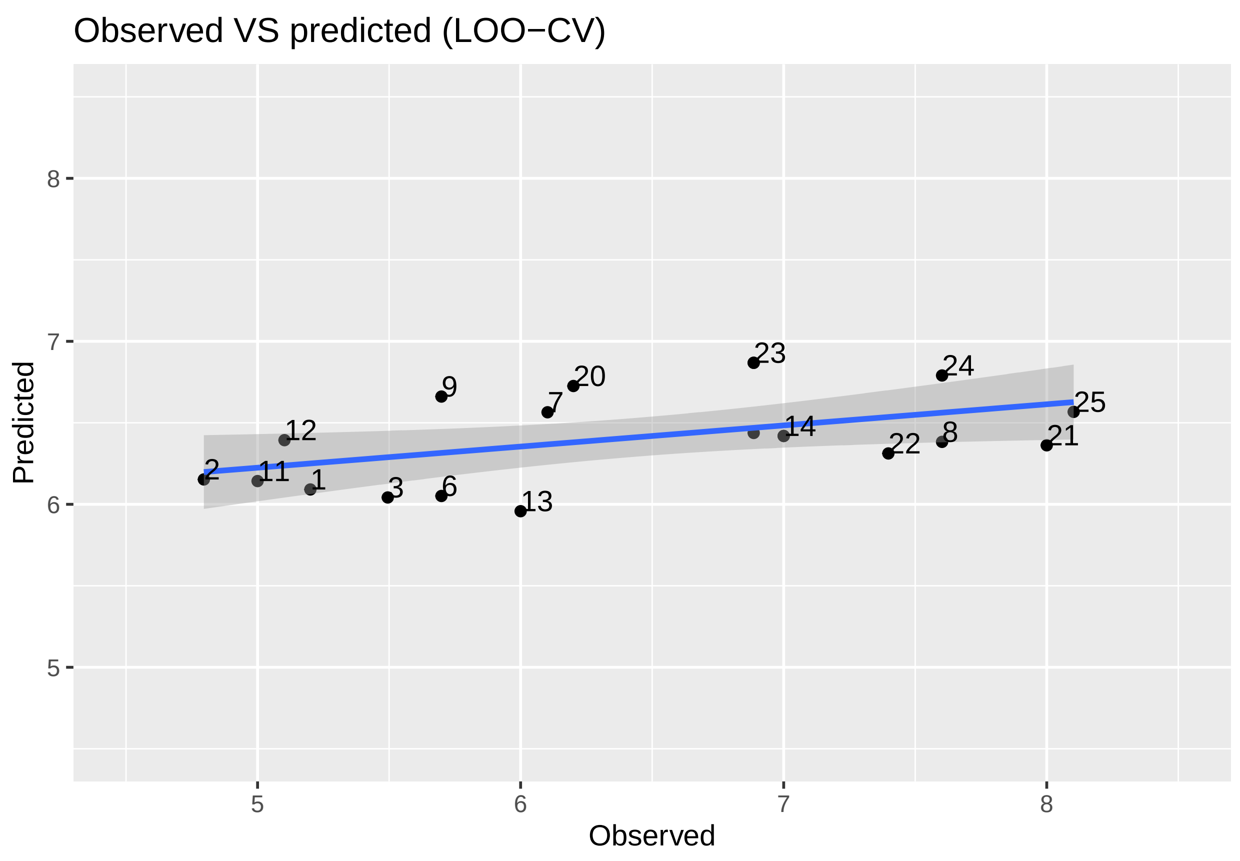
Calculating the metrics and plotting the results

```
## Unscaling observed and predicted data

Observado_LOO <- Observado_tr
Predicho_LOO <- Desescalar(Pred_CV)

LOO_df <- as.data.frame(cbind(Observado_LOO, Predicho_LOO))
rownames(LOO_df) <- Muestra[1:20]

## Metric calculation
LOO_M <- Metrica_LOO(Observado_LOO, Predicho_LOO)

LOO_M <- (do.call(rbind, LOO_M))
LOO_M <- round(LOO_M, 1)
print(LOO_M)
```

```
##          [,1]
## R2LOO     0.2
## RMSELOO   0.9
```

```
# Basic scatter plot LOO-CV
ggplot(LOO_df, aes(x = Observado_LOO, y = Predicho_LOO)) + geom_point() + geom_smooth(method = lm) +
    ggtitle("Observed VS predicted (LOO-CV)") + xlab("Observed") + ylab("Predicted") +
    geom_text(aes(label = rownames(LOO_df)), check_overlap = TRUE, hjust = 0, vjust = 0) +
    coord_cartesian(xlim = c(4.5, 8.5), ylim = c(4.5, 8.5))
```

```
## `geom_smooth()` using formula 'y ~ x'
```



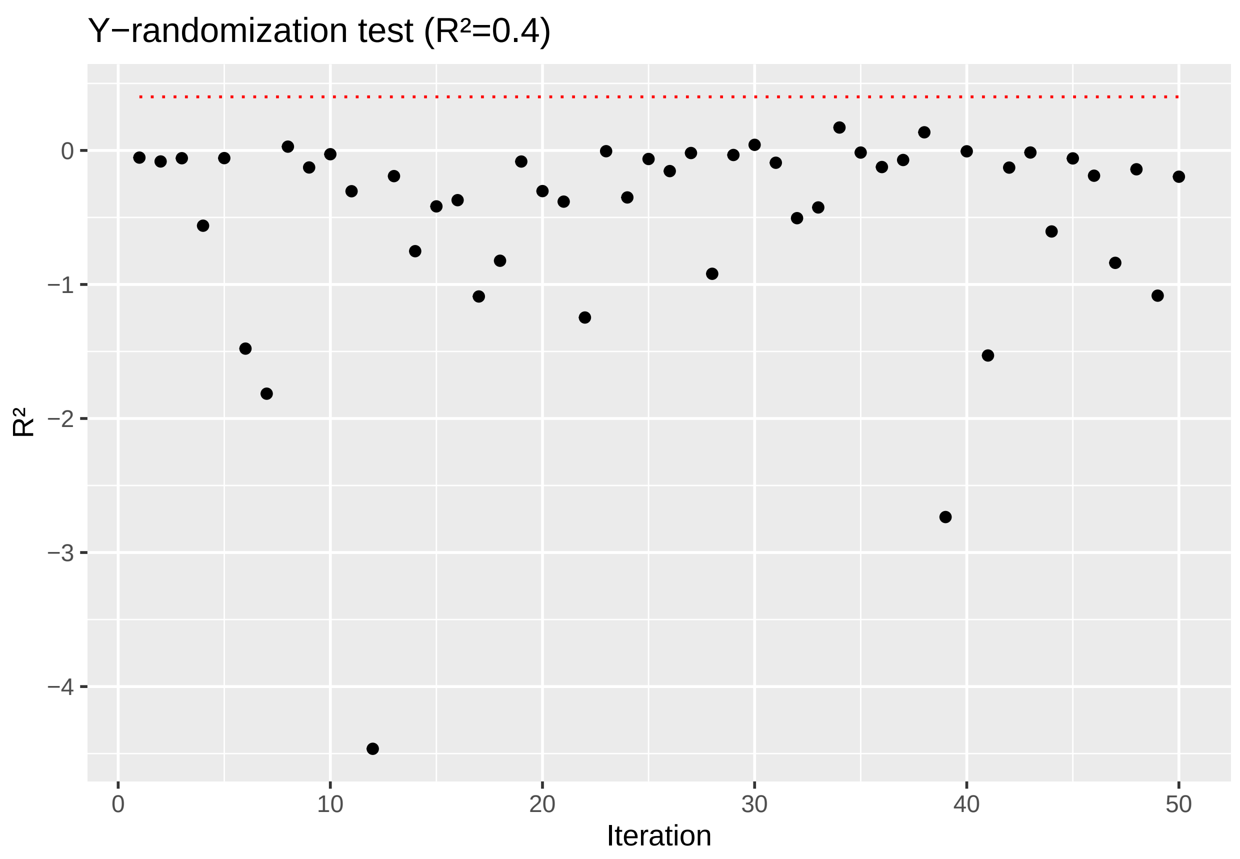
```r
ggsave("lassoloo.png")
```

```
## Saving 6.5 x 4.5 in image
## `geom_smooth()` using formula 'y ~ x'
```

Y-randomization test for LASSO model

Another statistical test to get insight about the reliability of the QSAR model is the Y-randomization. In this algorithm, the outcome variable (Y=pIC50) is randomized, and in each iteration a new model is build. The R2 for all the random models should be worse than the R2 from the constructed model. This test tries to prove that the model build does not come from random chance.

```r
## 50-fold loop for Y-randomization

## Empty list for store the data that comes from the loop

Score_Y <- list()
set.seed(1e+05)

for (i in 1:50) {

    ## Shuffling the outcome (pIC50), and building training and test sets with this
    ## randomized dataset
    Dataset_Y_shuffle <- as.matrix(Dataset_S)
    Y_shuffle <- sample(Dataset_S[, 275], replace = FALSE)
    Dataset_Y_shuffle[, 275] <- Y_shuffle
    Modelado_shuffle <- Dataset_Y_shuffle[Muestra, ]
    Prueba_shuffle <- Dataset_Y_shuffle[-Muestra, ]

    ## Lambda value getting from the first CV
    Mejor_Lambda_Y <- 0.4786

    ## Training the model with randomized outcome data

    Modelo_Y <- glmnet(Modelado_shuffle[, 1:274], Modelado_shuffle[, 275], alpha = 1,
        lambda = Mejor_Lambda_Y, standardize = FALSE)

    ## Predicting test set with randomized outcome data

    Pred_Y = predict.glmnet(Modelo_Y, s = Mejor_Lambda_Y, newx = Prueba_shuffle[,
        1:274])

    ## Unscaling data

    Observado_Y <- Desescalar(Prueba_shuffle[, 275])
    Predicho_Y <- Desescalar(Pred_Y)

    ## Model metrics
    Score_Y[[i]] <- Metrica_LOO(Observado_Y, Predicho_Y)
}

## Extracting metrics from the loop
Resul_Y_Random <- as.data.frame(t(unlist(Score_Y)))

par_indexes <- seq(2, 100, 2)
impar_indexes <- seq(1, 99, 2)

RMSEY <- Resul_Y_Random[, par_indexes]
Q2Y <- Resul_Y_Random[, impar_indexes]
Q2Y <- as.data.frame(t(Q2Y))
colnames(Q2Y) <- "R2"
rownames(Q2Y) <- c(1:50)
Q2Y$Iteration <- c(1:50)


library(ggplot2)

# Basic scatter plot Y-randomization
ggplot(Q2Y, aes(x = Iteration, y = R2)) + geom_point() + geom_line(aes(y = 0.4),
    linetype = "dotted", color = "red") + ggtitle("Y-randomization test (R²=0.4)") +
    ylab("R²")
```



```
ggsave("lassoyrandom.png")
```

```
## Saving 6.5 x 4.5 in image
```

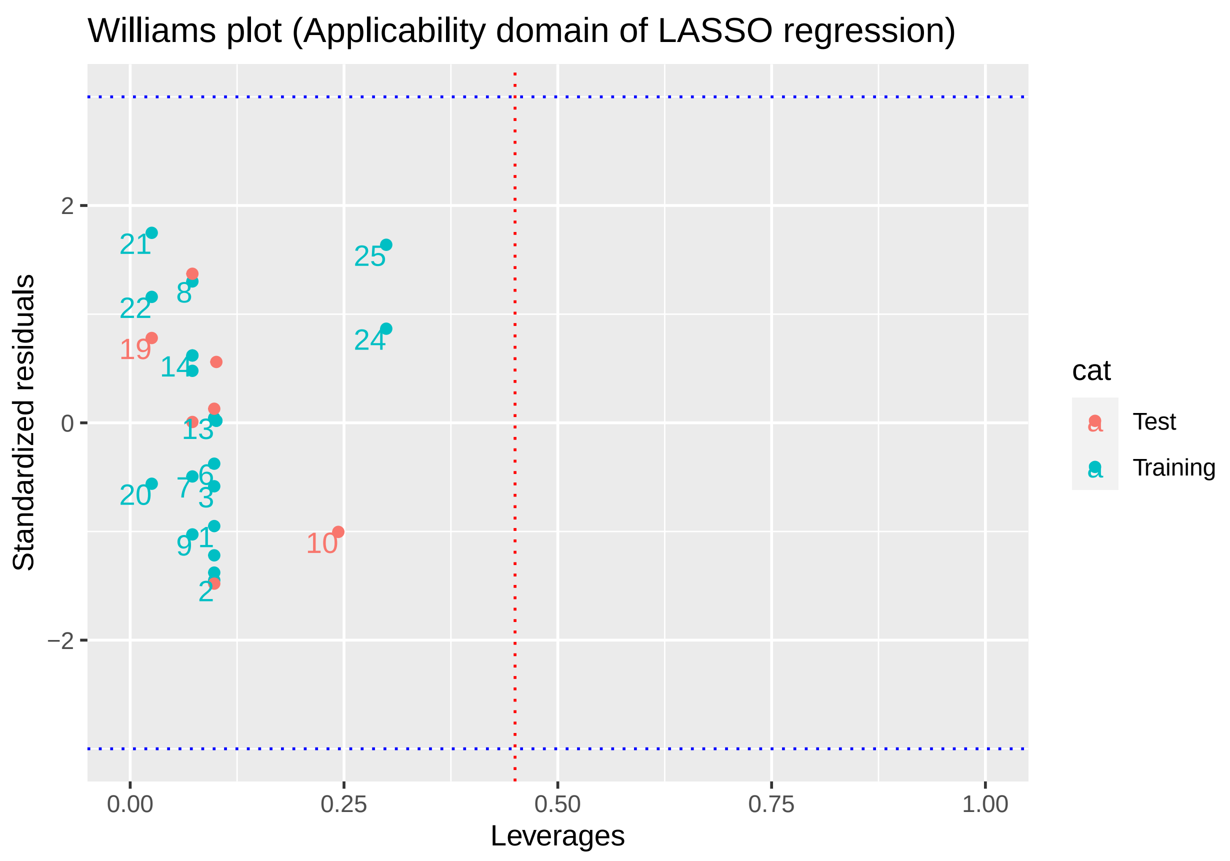
Since none of the randomized models had a better performance than the constructed model (R2=0.4), the QSAR is reliable.

Applicability domain (AD) of the QSAR model

The last step for the development of this QSAR model is the determination of the chemical space gives by the molecular descriptors used to construct the regression. The AD permits to interpolate in a chemical region with reliable results. Finally, the AD also detects possible outliers that are being extrapolated from the model. For more information about AD theory, please refer to the Manuscript.

```
### For training set

## Getting only columns with selected features by LASSO
library(dplyr)
X_AD <- as.data.frame(Modelamiento)
X_AD <- select(X_AD, F04.C.S., B10.C.S.)
X_AD <- as.matrix(X_AD)

## Mathematical calculation of leverages
H <- X_AD %*% solve(t(X_AD) %*% X_AD) %*% t(X_AD)  ##Hat matrix
hi <- diag(H)  ##Leverages

## Mathematical calculation of standardized residuals from LOO-CV prediction
```

```r
Residuales <- (Observado_LOO - Predicho_LOO)
K <- 20

## Standardized residuals
Stand_Residuales <- Residuales/sd(Residuales)

## Estimating warning leverage (every molecule that relies outside this limit,
## can't be interpolated and its predicted value is not trustworthy)

h_asterisco <- (3 * (2 + 1))/K  ##Two regressors and 20 observations h*=3(p+1)/n

### For test set

## Getting only columns with selected features by LASSO
X_test <- as.data.frame(Prueba)
X_test <- select(X_test, F04.C.S., B10.C.S.)
X_test <- as.matrix(X_test)

## Mathematical calculation of leverages
H_test <- X_test %*% solve(t(X_AD) %*% X_AD) %*% t(X_test)  ##Hat matrix for test
hi_test <- diag(H_test)  ##Leverages

## Mathematical calculation of standardized residuals from test prediction

Residuales_test <- (Observado_out - Predicho_out)
Stand_Residuales_test <- Residuales_test/sd(Residuales_test)

## Dataframes for training and test set of leverages and standardized residuals

Will_training <- as.data.frame(cbind(hi, Stand_Residuales))
colnames(Will_training) <- c("Leverages", "Standardized_residuals")
rownames(Will_training) <- Muestra[1:20]

Will_test <- as.data.frame(cbind(hi_test, Stand_Residuales_test))
colnames(Will_test) <- c("Leverages", "Standardized_residuals")
rownames(Will_test) <- c("4", "5", "10", "15", "16", "19", "26")

library(ggplot2)

Will_training$cat <- "Training"
Will_test$cat <- "Test"

ggplot_William <- rbind(Will_training, Will_test)

# Basic scatter plot Williams plot

ggplot(ggplot_William, aes(x = Leverages, y = Standardized_residuals, color = cat)) +
    geom_point() + xlim(0, 1) + geom_text(label = rownames(ggplot_William), check_overlap = TRUE,
    hjust = 1, vjust = 1) + geom_vline(xintercept = 0.45, linetype = "dotted", color = "red") +
    geom_hline(yintercept = c(-3, 3), linetype = "dotted", color = "blue") + ggtitle("Williams plot (Ap
    ylab("Standardized residuals")
```



```r
ggsave("lassoad.png")
```

With the Williams plot, the AD of the model is determined, and apparently all the training and test molecules are correctly interpolated into the chemical space defined by the molecular descriptors selected.

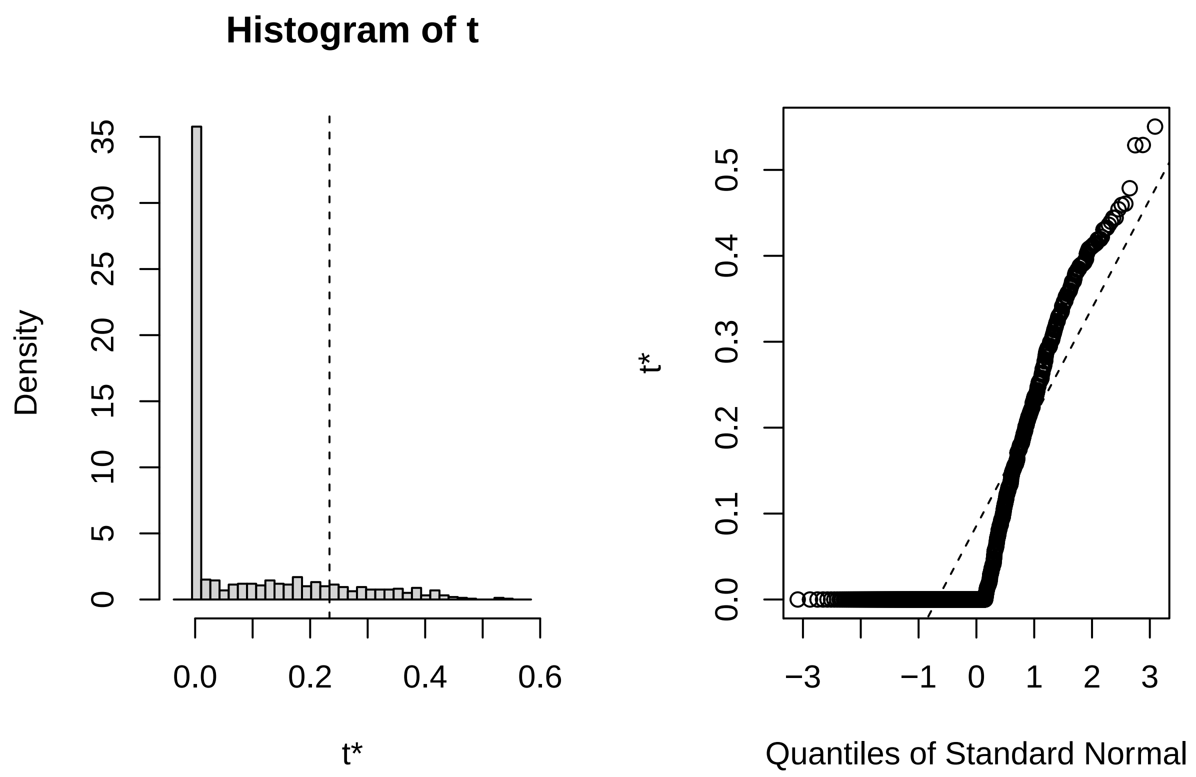
Bootstrapping the model to get standard errors and confidence intervals of the coefficients of regression

Since LASSO penalized regression is a technique that introduces a bias (λ) into the least square calculation in order to improve the variance modeling; it is not possible to get standard errors (SD) and confidence intervals (CI) as in a typical linear regression (that relies on the least square algorithm with a residual error normally distributed). So one needs to use bootstrapping sampling to replicate the experiment n times, and construct with it a distribution of the parameter (with the distribution, the standard error and confidence intervals are easily calculated). In this case, our interest is the SD and CI of the coefficient that multiplies the molecular descriptor (Please refer the equation presented above).

First coefficient SD and CI calculation

```r
## Defining a function that contains all the model
LASSO_Bootstrapping <- function(Conjunto_Datos, indices) {

    Predictores <- Conjunto_Datos[indices, 1:274]
    Respuesta <- Conjunto_Datos[indices, 275]

    ## Best lambda value

    Mejor_Lambda <- 0.4786
    ## Training LASSO model

    Mejor_Lasso <- glmnet(Predictores, Respuesta, alpha = 1, lambda = Mejor_Lambda,
        standardize = FALSE)

    ## Prediction LASSO model

    Predicho <- predict.glmnet(Mejor_Lasso, s = Mejor_Lambda, newx = Predictores)

    ## Regressors coefficients
    Coef_Import = coef(Mejor_Lasso)[, 1]
    Coef_Import <- as.data.frame(t(Coef_Import))

    salida <- c(Coef_Import$F04.C.S.)
    return(salida)
}

## Bootstrapping the model with a replicate value of 1000
library(boot)
Boot_LASSO_adj <- boot(Dataset_S, LASSO_Bootstrapping, R = 1000)
```

```
## Output of the bootstrapping sampling
plot(Boot_LASSO_adj)
```



```
Boot_LASSO_adj
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Dataset_S, statistic = LASSO_Bootstrapping, R = 1000)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.2336178 -0.1478848   0.1267312
```

```
## Getting CI for the coefficients
boot.ci(Boot_LASSO_adj, conf = 0.95)
```

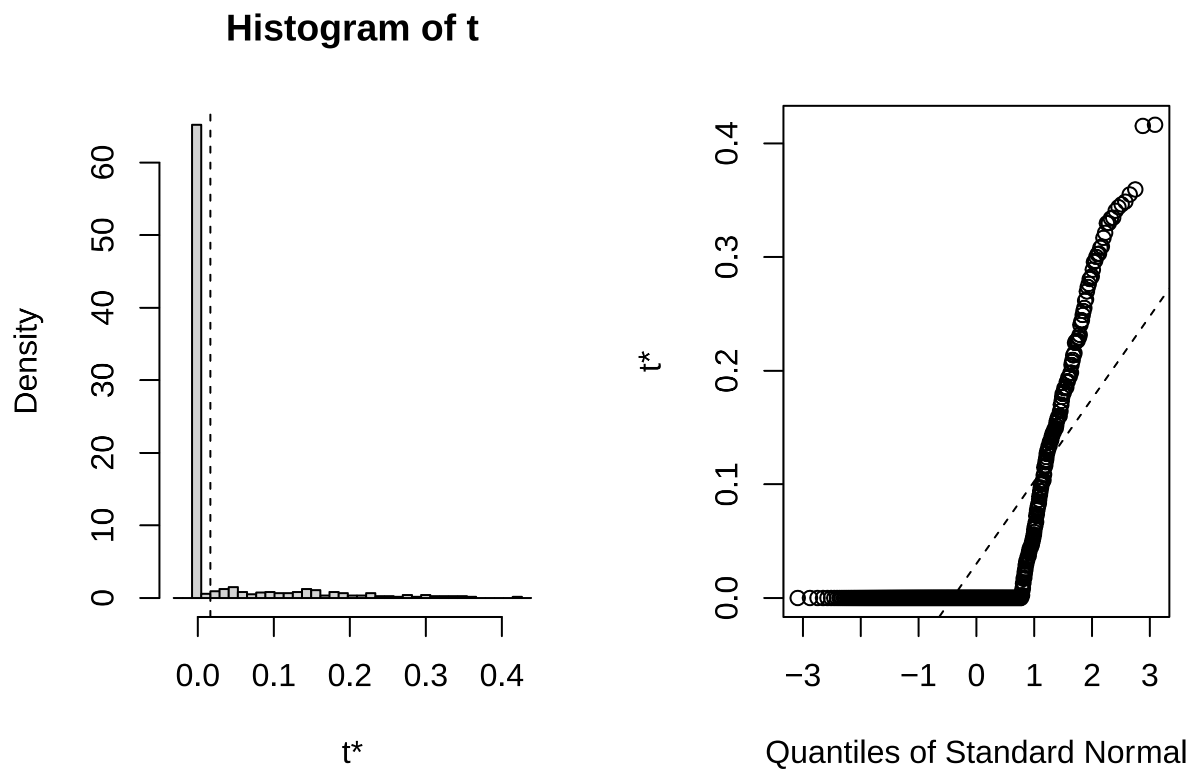
```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = Boot_LASSO_adj, conf = 0.95)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.1331,  0.6299 )   ( 0.0584,  0.4672 )
##
## Level     Percentile            BCa
## 95%   ( 0.0000,  0.4089 )   ( 0.0000,  0.5504 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

Second coefficient SD and CI calculation

```
## Defining a function that contains all the model

LASSO_Bootstrapping <- function(Conjunto_Datos, indices) {

    Predictores <- Conjunto_Datos[indices, 1:274]
    Respuesta <- Conjunto_Datos[indices, 275]

    ## Best lambda value

    Mejor_Lambda <- 0.4786

    ## Training LASSO model

    Mejor_Lasso <- glmnet(Predictores, Respuesta, alpha = 1, lambda = Mejor_Lambda,
        standardize = FALSE)

    ## Prediction LASSO model

    Predicho <- predict.glmnet(Mejor_Lasso, s = Mejor_Lambda, newx = Predictores)
```

```r
    ## Regressors coefficients
    Coef_Import = coef(Mejor_Lasso)[, 1]
    Coef_Import <- as.data.frame(t(Coef_Import))

    salida <- c(Coef_Import$B10.C.S.)
    return(salida)
}

## Bootstrapping the model with a replicate value of 1000
library(boot)
Boot_LASSO_adj <- boot(Dataset_S, LASSO_Bootstrapping, R = 1000)
```

```r
## Output of the bootstrapping sampling
plot(Boot_LASSO_adj)
```



```r
Boot_LASSO_adj
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Dataset_S, statistic = LASSO_Bootstrapping, R = 1000)
##
##
## Bootstrap Statistics :
```

```
##       original     bias    std. error
## t1* 0.01652973 0.01353916  0.07261946
```

```r
## Getting CI for the coefficients
boot.ci(Boot_LASSO_adj, conf = 0.95)
```

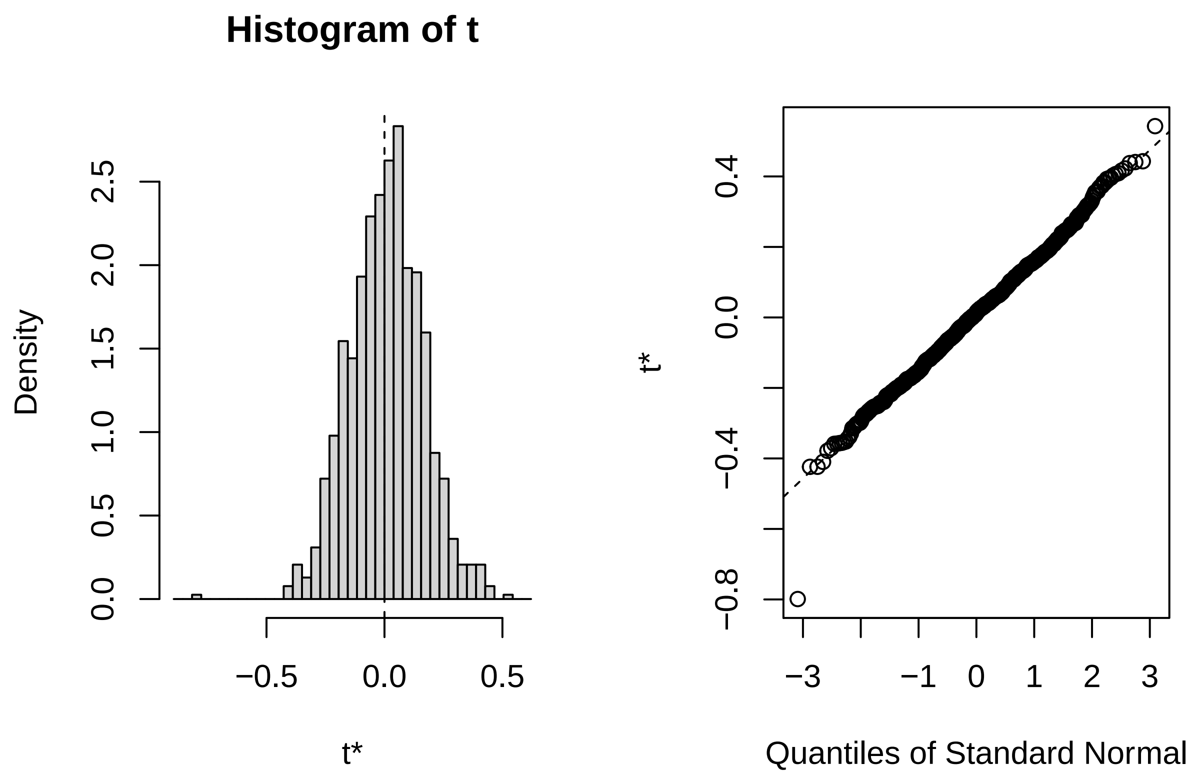
```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = Boot_LASSO_adj, conf = 0.95)
##
## Intervals :
## Level      Normal              Basic
## 95%   (-0.1393,  0.1453 )   (-0.2474,  0.0331 )
##
## Level     Percentile            BCa
## 95%   ( 0.0000,  0.2805 )   ( 0.0000,  0.4164 )
## Calculations and Intervals on Original Scale
## Warning : BCa Intervals used Extreme Quantiles
## Some BCa intervals may be unstable
```

Third coefficient SD and CI calculation

```r
## Defining a function that contains all the model

LASSO_Bootstrapping <- function(Conjunto_Datos, indices) {

    Predictores <- Conjunto_Datos[indices, 1:274]
    Respuesta <- Conjunto_Datos[indices, 275]

    ## Best lambda value

    Mejor_Lambda <- 0.4786

    ## Training LASSO model

    Mejor_Lasso <- glmnet(Predictores, Respuesta, alpha = 1, lambda = Mejor_Lambda,
        standardize = FALSE)

    ## Prediction LASSO model

    Predicho <- predict.glmnet(Mejor_Lasso, s = Mejor_Lambda, newx = Predictores)

    ## Regressors coefficients
    Coef_Import = coef(Mejor_Lasso)[, 1]
    Coef_Import <- as.data.frame(t(Coef_Import))

    library(data.table)
    setnames(Coef_Import, old = c("(Intercept)"), new = c("Intercept"))

    salida <- c(Coef_Import$Intercept)
    return(salida)
}
```

```r
## Bootstrapping the model with a replicate value of 1000
library(boot)
Boot_LASSO_adj <- boot(Dataset_S, LASSO_Bootstrapping, R = 1000)
```

```r
## Output of the bootstrapping sampling
plot(Boot_LASSO_adj)
```



```r
Boot_LASSO_adj
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Dataset_S, statistic = LASSO_Bootstrapping, R = 1000)
##
##
## Bootstrap Statistics :
##          original      bias    std. error
## t1* -3.229172e-16 0.009092806   0.1553126
```

```r
## Getting CI for the coefficients
boot.ci(Boot_LASSO_adj, conf = 0.95)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
```

```
## CALL :
## boot.ci(boot.out = Boot_LASSO_adj, conf = 0.95)
##
## Intervals :
## Level      Normal              Basic
## 95%   (-0.3135,  0.2953 )   (-0.3228,  0.2803 )
##
## Level     Percentile            BCa
## 95%   (-0.2803,  0.3228 )   (-0.3100,  0.2908 )
## Calculations and Intervals on Original Scale
```

```
library(knitr)

Mol_Descriptor <- c("F04.C.S.", "B10.C.S.", "Intercept")
Error <- as.numeric(c(0.1267312, 0.07261946, 0.1526878))
CImin <- as.numeric(c(0.1331, -0.1393, -0.316))
CImax <- as.numeric(c(0.6299, 0.1453, 0.2826))
Coefficient <- as.numeric(c(0.28912994, 0.06287162, 0.0107879))

Parameters_Boot_LASSO <- as.data.frame(cbind(Error, CImax, CImin, Coefficient))
round(Parameters_Boot_LASSO, 3)
```

```
##   Error CImax  CImin Coefficient
## 1 0.127 0.630  0.133       0.289
## 2 0.073 0.145 -0.139       0.063
## 3 0.153 0.283 -0.316       0.011
```

```
Parameters_Boot_LASSO <- as.data.frame(cbind(Mol_Descriptor, Parameters_Boot_LASSO))

colnames(Parameters_Boot_LASSO) <- c("Molecular descriptor", "Standard error", "Maximum confidence inte
    "Minimum confidence interval", "Coefficient")

kable(Parameters_Boot_LASSO)
```

Molecular descriptor	Standard error	Maximum confidence interval	Minimum confidence interval	Coefficient
F04.C.S.	0.1267312	0.6299	0.1331	0.2891299
B10.C.S.	0.0726195	0.1453	-0.1393	0.0628716
Intercept	0.1526878	0.2826	-0.3160	0.0107879

Now, the QSAR model is finished.

Virtual screening

After the development of the QSAR model; two databases of molecules were taken from **PubChem** and **ZINC**. This molecules were filter by two parameters: **Glucosamine-like molecules** and **Lipinski rule of five**.

The *Lipinski rule of five* is a empirical rule for druglikeness of molecules, the physicochemical parameters for this rule are:

- No more than 5 hydrogen bond donors.
- No more than 10 hydrogen bond acceptors.
- A molecular mass less than 500 g/mol.

- An octanol-water partition coefficient (Log P) that does not exceed 5.

The *glucosamine-like molecules* are mandatory to kept into the databases since the QSAR was development under glucosamine derivative molecules, and the chemical space is defined or interpolated under this scaffold.

After the filtering we got 155 unique molecules in SMILES format. The correct protonation state for each molecule was carried out by **Gypsum-DL software** with a pH between 7.0 and 7.2. The tautomer and isomer distribution for each molecule was carried out by the same software, and for each molecule was generated two (310) and three (465) possible tautomer/conformer and protonated states in order to get a better insight of all the chemical states on the dataset.

Finally, the datasets were uploaded into OCHEM web platform in order to calculate mechanistic interpretable 2D and 3D molecular descriptors (the same molecular descriptors used for the QSAR development).

Two variants

```r
## Invoking datasets
library(dplyr)

## Alvadescriptors
alvdesc_2 <- read.csv("alvdesc-2.csv", stringsAsFactors = FALSE)

## MOPAC
mopac_2 <- read.csv("mopac-2.csv", stringsAsFactors = FALSE)

## Joining both molecular descriptor datasets

mopac_2$SMILES <- NULL

VS_db <- cbind(alvdesc_2, mopac_2)
SMILES <- VS_db$SMILES
```

Cleaning datasets

```r
## Converting to numeric
VS_db <- apply(VS_db, 2, as.numeric)
VS_db <- as.data.frame(VS_db)
VS_db$SMILES <- NULL

## Scaling data
Mean_VS <- as.data.frame(t(Mean))
SD_VS <- as.data.frame(t(SD))
Mean_VS$pIC50 <- NULL
SD_VS$pIC50 <- NULL

VS_db <- select(VS_db, names(Mean_VS))

VS_db <- as.matrix(VS_db)
VS_db_s <- scale(VS_db, center = Mean_VS, scale = SD_VS)
```

Predicting activites

```r
Predicho_out_VS <- predict.glmnet(Mejor_Lasso, s = Mejor_Lambda, newx = VS_db_s)
Predicho_out_VS <- Desescalar(Predicho_out_VS)
ID <- c(1:310)
```

```
Predicho_out_VS <- cbind(ID, Predicho_out_VS)

colnames(Predicho_out_VS) <- c("Index", "pIC50")

## Prediction
Predicho_out_VS <- as.data.frame(Predicho_out_VS)
class(Predicho_out_VS)
```

```
## [1] "data.frame"
```

```
write.csv(Predicho_out_VS[order(Predicho_out_VS$pIC50, decreasing = TRUE), ], file = "VS_LASSO-2.csv",
    row.names = FALSE)
head(Predicho_out_VS)
```

```
##   Index    pIC50
## 1     1 5.997322
## 2     2 5.997322
## 3     3 5.997322
## 4     4 5.997322
## 5     5 5.997322
## 6     6 5.997322
```

Three variants

```
## Invoking datasets
library(dplyr)

## Alvadescriptors
alvdesc_3 <- read.csv("alvdesc-3.csv", stringsAsFactors = FALSE)

## MOPAC
mopac_3 <- read.csv("mopac-3.csv", stringsAsFactors = FALSE)

## Joining both molecular descriptor datasets

mopac_3$SMILES <- NULL

VS_db3 <- cbind(alvdesc_3, mopac_3)
SMILES3 <- VS_db3$SMILES
```

Cleaning datasets

```
## Converting to numeric
VS_db3 <- apply(VS_db3, 2, as.numeric)
VS_db3 <- as.data.frame(VS_db3)
VS_db3$SMILES <- NULL

## Scaling data
Mean_VS <- as.data.frame(t(Mean))
SD_VS <- as.data.frame(t(SD))
Mean_VS$pIC50 <- NULL
SD_VS$pIC50 <- NULL

VS_db3 <- select(VS_db3, names(Mean_VS))
```

```r
VS_db3 <- as.matrix(VS_db3)
VS_db3_s <- scale(VS_db3, center = Mean_VS, scale = SD_VS)
```

Predicting activites

```r
Predicho_out_VS3 <- predict.glmnet(Mejor_Lasso, s = Mejor_Lambda, newx = VS_db3_s)
Predicho_out_VS3 <- Desescalar(Predicho_out_VS3)
ID <- c(1:465)

Predicho_out_VS3 <- cbind(ID, Predicho_out_VS3)

colnames(Predicho_out_VS3) <- c("Index", "pIC50")

## Prediction
Predicho_out_VS3 <- as.data.frame(Predicho_out_VS3)
class(Predicho_out_VS3)
```

```
## [1] "data.frame"
```

```r
write.csv(Predicho_out_VS3[order(Predicho_out_VS3$pIC50, decreasing = TRUE), ], file = "VS_LASSO-3.csv"
    row.names = FALSE)
head(Predicho_out_VS3)
```

```
##   Index    pIC50
## 1     1 5.997322
## 2     2 5.997322
## 3     3 5.997322
## 4     4 5.997322
## 5     5 5.997322
## 6     6 5.997322
```