Proteochemometrics (PCM) with 'camb'
Chemistry **A**ware **M**odel **B**uilder

Isidro Cortes-Ciriano*[1,3] and Daniel Murrell†[2,3]

[1] *Unite de Bioinformatique Structurale, Institut Pasteur and CNRS UMR 3825, Structural Biology and Chemistry Department, 25-28, rue Dr. Roux, 75 724 Paris, France.*
[2] *Unilever Centre for Molecular Science Informatics, Department of Chemistry, University of Cambridge, Cambridge, United Kingdom.*
[3] *Equal contributors*

May 6, 2014

In the following sections, we present a pipeline to generate a Proteochemometric (PCM) model for mammal cyclooxigenase (COX) inhibitors. For futher details about PCM, the interested reader is referred to ref [1] and [2]. Firstly, we load the package and set the working directory:

```
library(camb)
# setwd('path_to_working_directory')
```

1 Compounds

1.1 Reading and Preprocessing

We proceed to read the compounds. Given that some smiles contain smarts patterns where the hash symbol is present, it is necessary to switch off the argument *comment.char* in order not to clip the smiles:

*isidrolauscher@gmail.com
†dsmurrell@gmail.com

```
smiles <- read.table("smiles_COX.smi", header = FALSE,
    comment.char = c(""))
```

The function *StandardiseMolecules* enables the depiction of molecular structures in a similar way. Moreover, the different arguments allow to control the maximum number of (i) fluorines, (ii) chlorines, (iii) bromines, and (iv) iodines a molecules can exhibit in order to be kept. If using the default value, *i.e.* "-1", all molecules will be kept irrespective of the number of fluorines, chlorines, bromines, and iodines. Inorgnaic molecules are removed if the argument "removed.inorganic" is set to "TRUE", which is the default value. Additionally, upper and lower limits for the molecular mass can be set with the arguments "min.mass.limit" and "max.mass.limit". The default value is "-1". Thus, all molecules will be kept irrespective of their molecular mass. The name of the file containing the chemical structures is input to the argument "structures.file".

```
StandardiseMolecules(structures.file="smiles_COX.smi",
standardised.file="standardised.sdf",
removed.file="removed.sdf",
output="standardisation_COX_info.csv",
remove.inorganic=TRUE,
fluorine.limit=-1,
chlorine.limit=-1,
bromine.limit=-1,
iodine.limit=-1,
min.mass.limit=-1, #suggested value 20
max.mass.limit=-1) #suggested value  900
```

The properties of all molecules and an index, in the column "kept", indicating which molecules were deleted, are written to the file indicated in the argument "output". In this case the file is `"standardisation_COX_info.csv"`. Those molecules that passed the filters are written to the file indicated in the argument "standardised.file". By contrast, molecules that did not pass these filters are written to the file indicated in the "removed.file" argument.

```
standardised_info <- read.table("standardisation_COX_info.csv",
    header = TRUE, sep = "\t")
```

Default values of the arguments of the function "StandardiseMolecules" are not stringent. In the present case, all molecules were kept, thus all values in the columns "kept" are equal to "1".
The properties of a ".sdf" file can be inspected witn the function:

```
ShowPropertiesSDF("standardised.sdf")
```

The values corresponding to an individual property can be accessed with the function "GetPropertySDF". Similarly, the function "GetPropertiesSDF" retrieves the information for all properties of a given ".sdf" file. A data.frame with all properties is returned. The number of molecules from which the information has to be retrieved can be indicated with the argument `"number_processed"`. The default value for this argument is `"-1"`, which indicates that the properties will be extracted for all molecules in the input file.

```
GetPropertySDF("standardised.sdf", property = "property_name",
    number_processed = 10)

## Error:  line 2 did not have 11 elements

all_properties <- GetPropertiesSDF("standardised.sdf", number_processed = 10)

## Error:  no lines available in input

head(all_properties)

## Error:  error in evaluating the argument 'x' in selecting a method for function
'head':  Error:  object 'all_properties' not found
```

1.2 PaDEL Descriptors

One and two-dimensional PaDEL[3] descriptors and fingerprints can be calculated with the function "GeneratePadelDescriptors":

```
descriptors_COX <- GeneratePadelDescriptors( +
  standardised.file="smiles_COX.smi",threads = 1)
descriptors <- RemoveStandardisedPrefix(descriptors)
#saveRDS(descriptors, file="Padel_COX.rds")
#descriptors <- readRDS("Padel_COX.rds")
```

Sometimes, some descriptors are not calculated for all molecules, thus giving a "NA" or "Inf" as descriptor value. Instead of removing that descriptor for all molecules, the missing descriptor values can be imputed from the corresponding descriptor values of the rest of molecules. Descriptor values equal to "Inf" are converted to "NA". For the imputation of missing dscriptor values, the R package *impute* is required. Depending on the R version, it can be accessed from either *CRAN* or *Bioconductor*.

```
descriptors <- ReplaceInfinitesWithNA(descriptors)
descriptors <- ImputeFeatures(descriptors)
```

1.3 Circular Morgan Fingerprints

The calculation of circular Morgan fingerprints requires the python library RDkit, given that the function "MorganFPs" calls a python script for the calculation of this type of fingerprints. For a detailed discussion about circular Morgan fingerprints, we refer the interested reader to ref. [4]. When using integrated development environments (IDE) such as RStudio, the environment variables might not be defined within the R session. However, they can be redefined with the R function "Sys.setenv". In any case, the function "MorganFPs" requires this information in the arguments "PythonPath", path to python in the system, and "RDkitPath", the path to the RDkit library. For instance, the information of the latter is contained in the environment variable `$RDBASE` in Mac OS.

```
Sys.setenv(RDBASE = "/usr/local/share/RDKit")
Sys.setenv(PYTHONPATH = "/usr/local/lib/python2.7/site-packages")
fps_COX_512 <- MorganFPs(bits = 512, radius = 2, type = "smi",
    mols = "smiles_COX.smi", output = "COX", keep = "hashed_counts",
    RDkitPath = "/usr/local/share/RDKit", PythonPath = "/usr/local/lib/python2.7/site-pa
    verbose = TRUE, images = FALSE, unhashed = FALSE,
    extFileExtension = FALSE, extMols = FALSE, unhashedExt = FALSE,
    logFile = FALSE)
saveRDS(fps_COX_512, file = "fps_COX_512.rds")
fps_COX_512 <- readRDS("fps_COX_512.rds")
```

In the following paragraph we describe in detail the arguments of the function "MorganFPs":

- **bits:** Number of bits of the hashed fingerprints. The default value is 512.

- **radius:** Maximum radius of the substructures. A radius of 2 is equivalent to `ECFP-4`, where 4 corresponds to the diameter. The default value is 2. More information on ECFP fingerprints can be found here: `http://www.chemaxon.com/jchem/doc/user/ECFP.html`.

- **type:** File format containing the input molecules. The default value is ".smi".

- **mols:** File containing the input molecules.

- **output:** Label that will be appended to all ouput files (see below).

- **keep:** The fingeprints that will be kept after the calculation. Apart from calculating different types of fingerprints, the function returns a data.frame with the type of fingerprints specified with this argument. Possible types are: `hashed_binary`,`hashed_counts`, `unhashed_binary`, `unhashed_counts`, and if applicable, `hashed_binaryEXT`, `hashed_countsEXT`, `unhashed_binaryEXT` and `unhashed_countsEXT`. The default value is "hashed$_b inary$".**images** :$IfTR$

- **unhashed:** If TRUE, unhashed fingeprints, both in binary format and with counts, are calculated. The default value is FALSE.

- **verbose:** If TRUE, information about the progression of the calculation is printed. The default value is FALSE.

- **RDkitPath:** The path to the folder containing the RDkit library in your computer. On mac, the environment variable `$RDBASE` contains this information. The default value is "/usr/local/share/RDKit".

- **PythonPath:** Path to python (`$PYTHONPATH`). The default value is "/usr/local/lib/python2.7/site-packages".

- **extFileExtension:** If not FALSE, file extension for the file containing the molecules for which unhashed fingeprints are to be calculated with respect to the pool of substructures in the molecules present in the file specified in "mols". The default value is FALSE.

- **extMols:** If not FALSE, file containing the molecules for which unhashed fingeprints are to be calculated with respect to the pool of substructures in the molecules present in the file specified in "mols". The default value is FALSE.

- **unhashedExt:** If TRUE, unhashed fingerprints are calcualted for the molecules specified in "extMols". The default value is FALSE.

- **logFile:** If not FALSE, file where the log messages will be dropped. The default value is FALSE.

add all options of the functions

2 Targets

2.1 Read and Preprocessing

We read the amino acids from a .csv file:

```
amino_compound_IDs <- read.table("AAs_COX.csv", sep = ",",
    header = TRUE, colClasses = c("character"), row.names = 1)
amino_compound_IDs <- amino_compound_IDs[, 2:ncol(amino_compound_IDs)]
```

Now, 5 Z-scales are calculated, which will serve to describe the target space in the PCM models. Subsequently, we save the descriptors to a ".rds" file.

```
amino_compound_IDs_zscales <- AA_descs(Data = amino_compound_IDs,
    type = "Z5")
saveRDS(amino_compound_IDs_zscales, file = "Z5_COX.rds")
```

```
amino_compound_IDs_zscales <- readRDS("Z5_COX.rds")
```

In the case that we needed whole sequence descriptors, they can be calculated with the function "SeqDescs". The function takes as argument either a UniProt identifier, or either a matrix or dataframe with the protein sequences. If a UniProt identifier is provided, the function gets firstly the sequence and then calculates the descriptors on the sequence.

```
Seq_descriptors_P00374 <- SeqDescs("P00374", UniProtID = TRUE,
    type = c("AAC", "DC"))
```

The available types of whole sequence descriptors are:[5]

- Amino Acid Composition ("AAC")
- Dipeptide Composition ("DC")

- Tripeptide Composition ("TC")
- Normalized Moreau-Broto Autocorrelation ("MoreauBroto")
- Moran Autocorrelation ("Moran")
- Geary Autocorrelation ("Geary")
- CTD (Composition/Transition/Distribution) ("CTD")
- Conjoint Traid ("CTriad")
- Sequence Order Coupling Number ("SOCN")
- Quasi-sequence Order Descriptors ("QSO")
- Pseudo Amino Acid Composition ("PACC")
- Amphiphilic Pseudo Amino Acid Composition ("APAAC")

2.2 Reading the Data-set Information

Now, we are going to read the file with the information about the dataset, namely: target names, bioctivities, etc.. Note that when reading smiles from a ".csv" file into an R dataframe, the smiles are clipped after a hash ("#") symbol. A good practice is thus to also keep the smiles alone in a {.smi,.smiles} file.

```
dataset <- readRDS("COX_dataset_info.rds")
bioactivity <- dataset$standard_value
```

The bioactivity is in nM. We convert it to pIC50. To do that, we multiply by `10^-9` to convert the bioactivity units to M. Subsequently, the negative logarithm to base 10 is calculated:

```
bioactivity <- bioactivity * 10^-9
bioactivity <- -log(bioactivity, base = 10)
```

3 Data-set Visualization

Compounds can be depicted with the function "PlotMolecules". This function returns a list of four plots. Additionally, plots can also be written into a ".pdf" file if the argument "pdf.file" is not NULL. The argument "IDs" corresponds to the index of the molecules in the input file which are to be depicted. The name of the molecule in the input file will be used as the title of the image if the argument "useNameAsTitle" is set to TRUE.

```
plot_molecules <- PlotMolecules("standardised.sdf",
    IDs = c(1, 2, 3, 4), pdf.file = NULL, useNameAsTitle = TRUE,
    PDFMain = NULL)

## [1] 1
## [1] 2
## [1] 3
## [1] 4

print(plot_molecules[[1]])
```

The distribution of the response variable can be explored with the function "DensityResponse" in the following way: variable:

```
dens_resp <- DensityResponse(bioactivity, xlab = "pIC50",
    main = "", ylab = "Densitiy", TitleSize = 30, XAxisSize = 22,
    YAxisSize = 22, TitleAxesSize = 24, AngleLab = 0,
    lmar = 0, rmar = 0, bmar = 0, tmar = 0, binwidth = 0.3)
```

Plotting a PCA analysis of the target descriptors gives:

```
target_PCA <- PCAProt(amino_compound_IDs_zscales, SeqsName = dataset$accession)
plot_PCA_Cox <- PCAProtPlot(target_PCA, PointSize = 10,
    main = "", TitleSize = 30, XAxisSize = 20, YAxisSize = 20,
    TitleAxesSize = 28, LegendPosition = "bottom",
    RowLegend = 3, ColLegend = 5, LegendTitleSize = 15,
    LegendTextSize = 15)
```

incluir en PCAprotta aa = summary(ana) aa*importance*

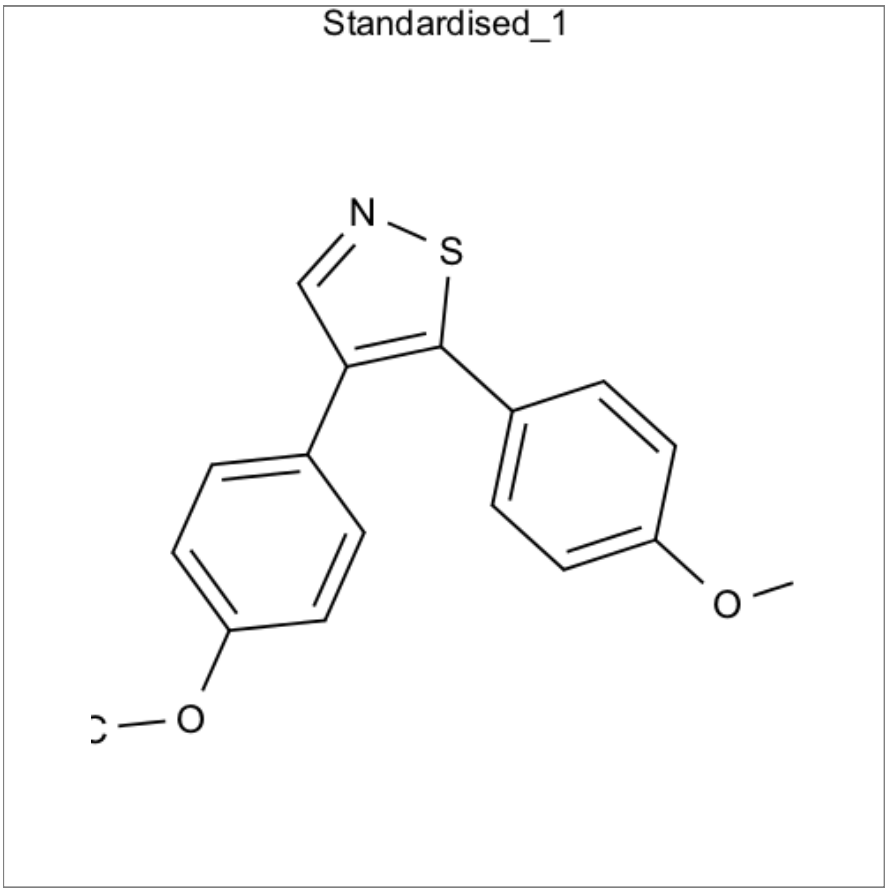

Figure 1: Example of compoud depiction.

aa*sdev*

incluir tanimoto

Similarly, we can analyze the chemical space by calculating pairwise compound similarities based upon the compound descriptors. In this case, we use the Jaccard metric to calculate the distance between compounds.

```
pw_dist_comp_fps <- PairwiseDist(fps_COX_512, method = "jaccard")
saveRDS(pw_dist_comp_fps, file = "pairwise_dist_COX.rds")
```

```
pw_dist_comp_fps <- readRDS("pairwise_dist_COX.rds")
plot_pwd <- PairwiseDistPlot(pw_dist_comp_fps, xlab = "Jaccard Similarity",
    ylab = "Density", TitleSize = 26, XAxisSize = 20,
    YAxisSize = 20, TitleAxesSize = 24, lmar = 0, rmar = 0,
    bmar = 0, tmar = 0, AngleLab = 0)
```

```
grid.arrange(dens_resp, plot_pwd, nrow = 2)

## Error:  object 'dens_resp' not found
```

Before any modeling attempt, it is interesting to know which is the maximum performance achievable *on the basis* of the available data.

By that, we consider the experimental uncertainty and the size of our data-set. In this case, a Gaussian Process (GP) model was trained in Matlab (data not shown) where the experimental uncertainty was optimized as a hyperparameter. The obtained value was 0.60. This value is in accordance with recently published value of 0.68 for public IC50 data. With the function 'MaxPerf', we can calculate the maximum achievable performance:

```
max_performance <- MaxPerf(meanNoise = 0, sdNoise = 0.6,
    meanResp = mean(bioactivity), sdResp = sd(bioactivity),
    lenPred = 800)
```

The function returns a list of four plots. By using the function 'plotGrid' we can create a grid of plots in the following way:

```
plotGrid(plots = c(max_performance$p1, max_performance$p2,
    max_performance$p3, max_performance$p4))
```

4 Statistical Pre-processing

Bioactivity annotations in ChEMBL are sometimes redundant, meaning that for a given target-compound combination there are more than one annotated values.
To avoid this issue, we will remove redundant pairs and will keep the mean bioactivity value for those compound-target combinations repeated.

```
source("remove_duplicates.R")
```

Now, we load the dataset without repetitions generated in the previous step. In addition, we remove those columns not containing descriptors (e.g. compound name):

```
dataset <- readRDS("Whole_dataset_NO_REP.rds")
killset <- expression(c(tid, pref_name, accession,
    organism, chembl_id, standard_value, standard_units,
    standard_type, chembl_id.1, Name, Name.1, Name.2,
    rows))
bioactivity <- dataset$standard_value
compound_IDs <- dataset$chembl_id.1
dataset <- subset(dataset, select = -eval(killset))
```

Subsequently, we split the dataset into training (70%) and hold-out (external; 30%) sets that will be used to assess the predictive ability of the models. Furthermore, we remove the following descriptors: (i) those with a variance close to zero (near-zero variance), and (ii) those highly correlated:

```
# Split the dataset into a training and holdout set
dataset <- SplitSet(compound_IDs, dataset, bioactivity,
    percentage = 30)
```

```
# Remove the descriptors that are highly correlated
# or have low variance
dataset <- RemoveNearZeroVarianceFeatures(dataset,
    frequencyCutoff = 30/1)
dataset <- RemoveHighlyCorrelatedFeatures(dataset)
```

We convert the descriptors to z-scores by centering them to zero mean and scaling their values to unit variance:

```
dataset <- PreProcess(dataset)
```

Given that cross-validation (CV) will be used to optimize the hyperparameters of the models, we divide the training set in 5 folds:

```
dataset <- GetCVTrainControl(dataset, seed = 1, folds = 5,
    repeats = 1, returnResamp = "none", returnData = FALSE,
    savePredictions = TRUE, verboseIter = TRUE, allowParallel = TRUE,
    index = createMultiFolds(y.train, k = folds, times = repeats))
saveRDS(dataset, file = "dataset_COX_preprocessed.rda")
```

All models are trained with the same CV options, *i.e.* hte arguments of the function 'GetCV-TrainControl' to allow ensemble modeling (see below).

5 Model Training

```
dataset <- readRDS("dataset_COX_preprocessed.rda")
# Number of cores to be used during model training
cores <- 3
registerDoMC(cores)
```

5.1 Support Vector Machines (SVM)

Firstly, a SVM will be trained[6]. We define an exponential grid (base 2) to optimize the hyperparameters:

```
method <- "svmRadial"
exp_grid <- expGrid(power.from = -8, power.to = -6,
    power.by = 2, base = 2)
tune.grid <- expand.grid(.sigma = exp_grid)
```

Training:

```
modelCoxSVMrad <- train(dataset$x.train, dataset$y.train,
    method, tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(modelCoxSVMrad, file = "model_SVM.rds")
```

5.2 Random Forest

We proceed similarly in the case of a random forest (RF) model[7].

```
method <- "rf"

modelCoxRF <- train(dataset$x.train, dataset$y.train,
    method, trControl = dataset$trControl)
saveRDS(modelCoxRF, file = "model_RF.rds")
```

Loading the RF model.

```
modelCoxRF <- readRDS("model_RF.rds")
```

5.3 Gradient Boosting Machine

We proceed similarly in the case of a gradient boosting machine (GBM) model[8].

```
method <- "gbm"
tune.grid <- expand.grid(.shrinkage = c(0.04, 0.08,
    0.12, 0.16), .n.trees = c(500), .interaction.depth = c(25))
modelCoxGBM <- train(dataset$x.train, dataset$y.train,
    method, tuneGrid = tune.grid, trControl = dataset$trControl)
saveRDS(modelCoxGBM, file = "model_GBM.rds")
```

6 Model Evaluation

Once the models are trained, the cross validated metrics can be calculated: We assume the metric used for the choice of the best combination of hyperparameters is 'RMSE'. In the following we focus on the RF model, though the same can be applied to the GBM and SVM models.

```
RMSE_CV_rf = signif(min(as.vector(na.omit(modelCoxRF$results$RMSE))),
    digits = 3)
Rsquared_CV_rf = modelCoxRF$results$Rsquared[which(modelCoxRF$results$RMSE %in%
    min(modelCoxRF$results$RMSE, na.rm = TRUE))]
print(RMSE_CV_rf)

## [1] 0.775

print(Rsquared_CV_rf)

## [1] 0.5914
```

On the basis of the soundness of the obtained models, we predict the values for the external (hold-out) set:

```
holdout.predictions <- as.vector(predict(modelCoxRF$finalModel,
    newdata = dataset$x.holdout))
```

We evaluate the predictive ability of our models by calculation the following statistical metrics:

Internal validation:

$$q^2_{int} = 1 - \frac{\sum_{i=1}^{N}(y_i - \widetilde{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_{tr})^2} \tag{1}$$

$$RMSE_{int} = \frac{\sqrt{(y_i - \widetilde{y}_i)^2}}{N} \tag{2}$$

where N, y_i, $\widetilde{y}_i$ and $\bar{y}_{tr}$ represent the size of the training set, the observed, the predicted and the averaged values of the response variable for those datapoints included in the training set. The ith position within the training set is defined by i.

External validation:

$$q^2_{ext} = 1 - \frac{\sum_{j=1}^{N}(y_j - \widetilde{y}_j)^2}{\sum_{j=1}^{N}(y_j - \bar{y}_{ext})^2} \tag{3}$$

$$RMSE_{ext} = \frac{\sqrt{(y_i - \widetilde{y}_i)^2}}{N} \tag{4}$$

$$R^2_{ext} = \frac{\sum_{i=1}^{N}(y_i - \bar{y}_{ext})(\widetilde{y}_i - \bar{\widetilde{y}}_{ext})}{\sqrt{\sum_{i=1}^{N}(y_i - \bar{y}_{ext})^2 \sum (\widetilde{y}_i - \bar{\widetilde{y}}_{ext})^2}} \tag{5}$$

$$R^2_{0\,ext} = 1 - \frac{\sum_{j=1}^{N}(y_j - \widetilde{y}^{r0}_j)^2}{\sum_{j=1}^{N}(y_j - \bar{y}_{ext})^2} \tag{6}$$

where N, y_j, $\widetilde{y}_j$, $\bar{y}_{ext}$ and $\breve{y}_j$ represent the size of the training set, the observed, the predicted, and the averaged values of the response variable for those datapoints comprising the external set. The jth position within the external set is defined by j. $R^2_{0\,ext}$ is the square of the coefficient of determination through the origin, being $\widetilde{y}^{r0}_j = k\widetilde{y}_j$ the regression through the origin (observed versus predicted) and k its slope.
For a detailed discussion of both the evaluation of the predictive ability through the external set and different formulations for q^2, see ref.[9]. To be considered as predictive, a model must satisfy the following criteria:[10, 11]

1. $q^2_{int} > 0.5$
2. $R^2_{ext} > 0.6$
3. $\frac{(R^2_{ext} - R^2_{0\,ext})}{R^2_{ext}} < 0.1$
4. $0.85 \leq k \leq 1.15$

The metrics for the external validation are given by:

```
MetricsRf <- Validation(holdout.predictions, dataset$y.holdout)
```

To have a look at the correlation between predicted and observed values, we can use the 'CorrelationPlot' function:

```
CorrelationPlot(pred = holdout.predictions, obs = dataset$y.holdout,
    PointSize = 3, ColMargin = "blue", TitleSize = 26,
    XAxisSize = 20, YAxisSize = 20, TitleAxesSize = 24,
    margin = 2, PointColor = "black", PointShape = 16,
    MarginWidth = 1, AngleLab = 0, xlab = "Observed",
    ylab = "Predicted")
```

7 Ensemble Modeling

In the following section, we applied two ensemble modeling techniques, namely greedy optimization and model stacking, to create ensembles of models. Further information can be found in ref [12] and [13].
To get (i) the training set, (ii) the external set, (iii) the transformation applied to center and scale the descriptors before model training, and (iv) the model training options, we run the following lines:

```
data <- list()
attach(dataset)
data$transformation <- transformation
data$x.train <- x.train
data$y.train <- y.train
data$x.test <- x.holdout  # external set
data$y.test <- y.holdout  # external set
data$trControl <- trControl
saveRDS(data, file = "data_ensemble.rds")
detach(dataset)
```

Subsequently, we load the models previously trained. The list of models is in the file modelsEnsemble. Once all models have been loaded, we create the following ensemble:

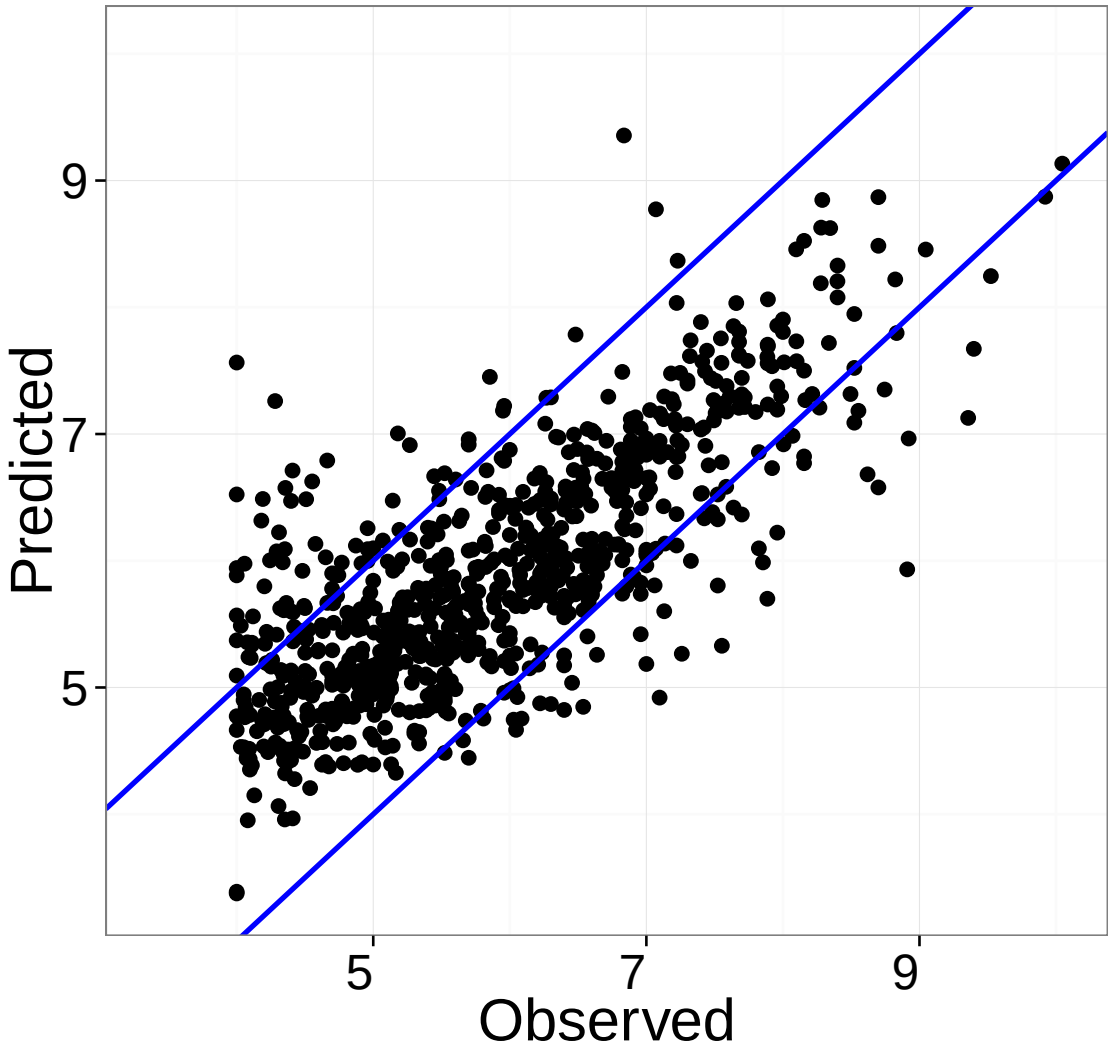

Figure 2: Observed vs Predicted

```
greedy <- caretEnsemble(all.models, iter = 1000L)
sort(greedy$weights, decreasing = TRUE)

all.models <- list()
models <- as.vector(read.table("modelsEnsemble")$V1)

for (i in 1:length(models)) {
    model_load = paste("readRDS('", models[i], "')",
        sep = "")
    assign(paste("model_", i, sep = ""), eval(parse(text = model_load)))
    all.models[[length(all.models) + 1]] <- eval(parse(text = paste("model_",
        i, sep = "")))
}

names(all.models) <- sapply(all.models, function(x) x$method)
sort(sapply(all.models, function(x) min(as.vector(na.omit(x$results$RMSE)))))
```

Once all models have been loaded, we create the following ensemble:

```
greedy <- caretEnsemble(all.models, iter = 1000L)
sort(greedy$weights, decreasing = TRUE)
```

```
  # make a linear regression ensemble
  linear <- caretStack(all.models, method='glm')
summary(linear$ens_model$finalModel)

# make Elastic Net ensemble
enet_ens <- caretStack(all.models, method='enet')
coefs_enet_ens <- enet_ens$ens_model$finalModel$beta.pure +
[ncol(enet_ens$ens_model$finalModel$beta.pure)+1,]

# make SVM linear ensemble
trControl <- trainControl(method = "cv",  number=5)
tune.grid <- expand.grid(.C=expGrid(power.from=-14,
```

```
                                        power.to=10,power.by=1,base=2))
linear_svm <- caretStack(all.models, method='svmLinear',
                         trControl=trControl,tuneGrid=tune.grid)

# make SVM radial ensemble
trControl <- trainControl(method = "cv",  number=5)
tune.grid <- expand.grid(.sigma=expGrid(power.from=-14,
                                        power.to=10,power.by=1,base=2),
                         .C=expGrid(power.from=-14,power.to=10,
                                    power.by=2,base=2))
radial_svm <- caretStack(all.models, method='svmRadial',
                         trControl=trControl,tuneGrid=tune.grid)
```

We proceed to predict the bioactivities for the external (hold-out) set,

```
preds <- data.frame(sapply(all.models, predict, newdata=dataa$x.test))
preds$ENS_greedy <- predict(greedy, newdata=dataa$x.test)
preds$ENS_linear <- predict(linear, newdata=dataa$x.test)
preds$ENS_enet <- predict(enet_ens, newdata=x.test)
preds$ENS_SVMrad <- predict(radial_svm, newdata=x.test)c
preds$ENS_SVMlin <- predict(linear_svm, newdata=x.test)
```

and we calculate the metrics:

```
# Calculate metrics (We could also have applied
# Validation instead.)
Q2s <- apply(preds, 2, function(x) Qsquared(x, dataa$y.test))
R2s <- apply(preds, 2, function(x) Rsquared(x, dataa$y.test))
R20s <- apply(preds, 2, function(x) Rsquared0(x, dataa$y.test))
RMSEs <- apply(preds, 2, function(x) RMSE(x, dataa$y.test))
```

References

[1] G. J. P. van Westen, J. K. Wegner, A. P. IJzerman, H. W. T. van Vlijmen, and A. Bender, "Proteochemometric modeling as a tool to design selective compounds and for extrapolating to novel targets," *Med. Chem. Commun.*, vol. 2, pp. 16–30, 1 2011.

[2] I. Cortes Ciriano, Q. U. Ain, V. Subramanian, E. B. Lenselink, O. Mendez Lucio, A. P. IJzerman, G. Wohlfahrt, P. Prusis, T. Malliavin, G. J. van Westen, and A. Bender, "Polypharmacology modelling using proteochemometrics: recent developments and future prospects," *In revision at Med. Chem. Comm*,

[3] C. W. Yap, "PaDEL-descriptor: an open source software to calculate molecular descriptors and fingerprints.," *J. Comput. Chem.*, vol. 32, pp. 1466–1474, 2011. [Online]. Available: `http://www.ncbi.nlm.nih.gov/pubmed/21425294`.

[4] D. Rogers and M. Hahn, "Extended-connectivity fingerprints.," *J. Chem. Inf. Model.*, vol. 50, no. 5, pp. 742–754, 2010.

[5] N. Xiao and Q. Xu, *Protr: protein sequence descriptor calculation and similarity computation with r*, R package version 0.2-1, 2014.

[6] A. Ben-Hur, C. S. Ong, S. Sonnenburg, B. Schlkopf, and G. Rtsch, "Support Vector Machines and Kernels for Computational Biology," *PLoS Computational Biology*, vol. 4, no. 10, F. Lewitter, Ed., e1000173, Oct. 2008.

[7] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[8] J. H. Friedman, "Greedy function approximation: a gradient boosting machine.," *The Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, Mathematical Reviews number (MathSciNet) MR1873328, Zentralblatt MATH identifier01829052.

[9] V. Consonni, D. Ballabio, and R. Todeschini, "Evaluation of model predictive ability by external validation techniques," *J. Chemometrics*, vol. 24, no. 3-4, pp. 194–201, 2010.

[10] A. Golbraikh and A. Tropsha, "Beware of q2!," *J. Mol. Graph. Model.*, vol. 20, no. 4, pp. 269–276, Jan. 2002.

[11] A. Tropsha, P. Gramatica, and V. K. Gombar, "The Importance of Being Earnest: Validation is the Absolute Essential for Successful Application and Interpretation of QSPR Models," *QSAR Comb. Sci.*, vol. 22, no. 1, pp. 69–77, 2003.

[12] Z. Mayer, "CaretEnsemble: framework for combining caret models into ensembles. [r package version 1.0]," 2013.

[13] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes, "Ensemble selection from libraries of models," in *Proceedings of the Twenty-first International Conference on Machine Learning*, ser. ICML '04, New York, NY, USA: ACM, 2004, p. 18.