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*

April 1, 2014

In the following sections, we present a pipeline to generate a whole Proteochemometric (PCM) pipeline. 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/camb/examples/COX')
```

1 Compounds

1.1 Reading and Preprocessing

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

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

Given that some smiles contain smarts patterns where the hash symbol is present, we need to avoid switch of the argumment comment.char in order not to clip the smiles:

```
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 the index (in the column 'kept') indicating which molecules were deleted are written to a file called standardisation_COX_info.csv.

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

In this case, the criteria we chose to remove compounds (the arguments of the StandardiseMolecules function) were not really stringent, so all molecules were kept. This can be seen in the kept field, given that for all compounds (rows) we have a value equal to 1.
Similarly, the properties of a .sdf file can be accessed witn the function:

```
ShowPropertiesSDF("standardised.sdf", type = 1)
```

A given property can be accessed, or all of them. In the latter case, a data.frame with all properties is returned:

```
GetPropertySDF("standardised.sdf", property = "property_name",
    number_processed = 10, type = 1)
all_properties <- GetPropertiesSDF("standardised.sdf", number_processed = 10,
    type = 1)
head(all_properties)
```

1.2 PaDEL Descriptors

```
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 calcualted for all molecules, thus giving a 'NA' or 'Inf' as descriptor values. 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. To do that, 'Inf' values are converted to 'NA', and then imputed. 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 python library RDkit is required since the function 'MorganFPs' uses a python script for the calculation. When using integrated development environments (IDE) such as RStudio, the environment variables might not be seen. We can redefine them with the R function 'Sys.setenv'. In any case, the function 'MorganFPs' requires this information in the arguments 'PythonPath' and 'RDkitPath'.

```
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)
saveRDS(fps_COX_512, file = "fps_COX_512.rds")
fps_COX_512 <- readRDS("fps_COX_512.rds")
```

2 Targets

2.1 Read and Preprocessing

We read the 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 calculates, which will serve to describe the target space in the machine learning models. Ensuingly, we save the descriptors in 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 functin 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:[3]

- 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.. Be careful: when reading smiles from a .csv file into an R dataframe, the smiles are clipped after a hash ('#') symbol. Good practice: 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:

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

3 Data-set Visualization

Compounds can be depicted with the function 'PlotMolecules'. It returns a list of four plots, and plots can also be written into a .pdf file.

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

We can have a look at the response 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)
```

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.

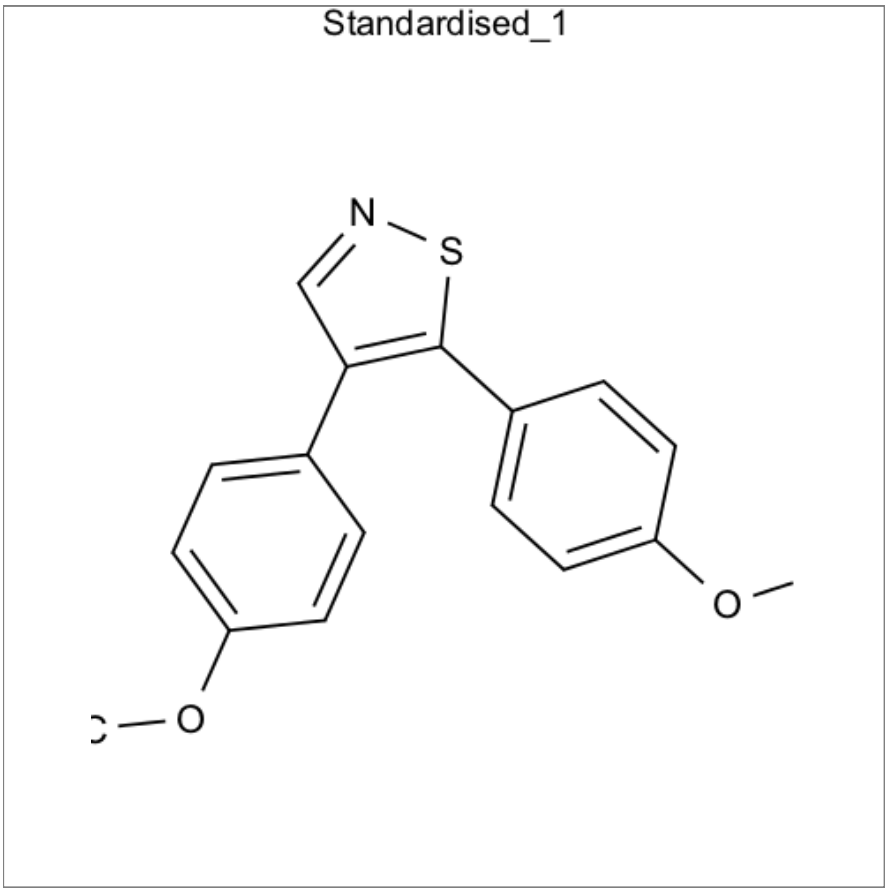

Figure 1: Example of compoud depiction.

```
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)
```

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 prublished value of 0.68 for public IC50 data. With the function 'MaxPerf', we can calculate the manximum 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.

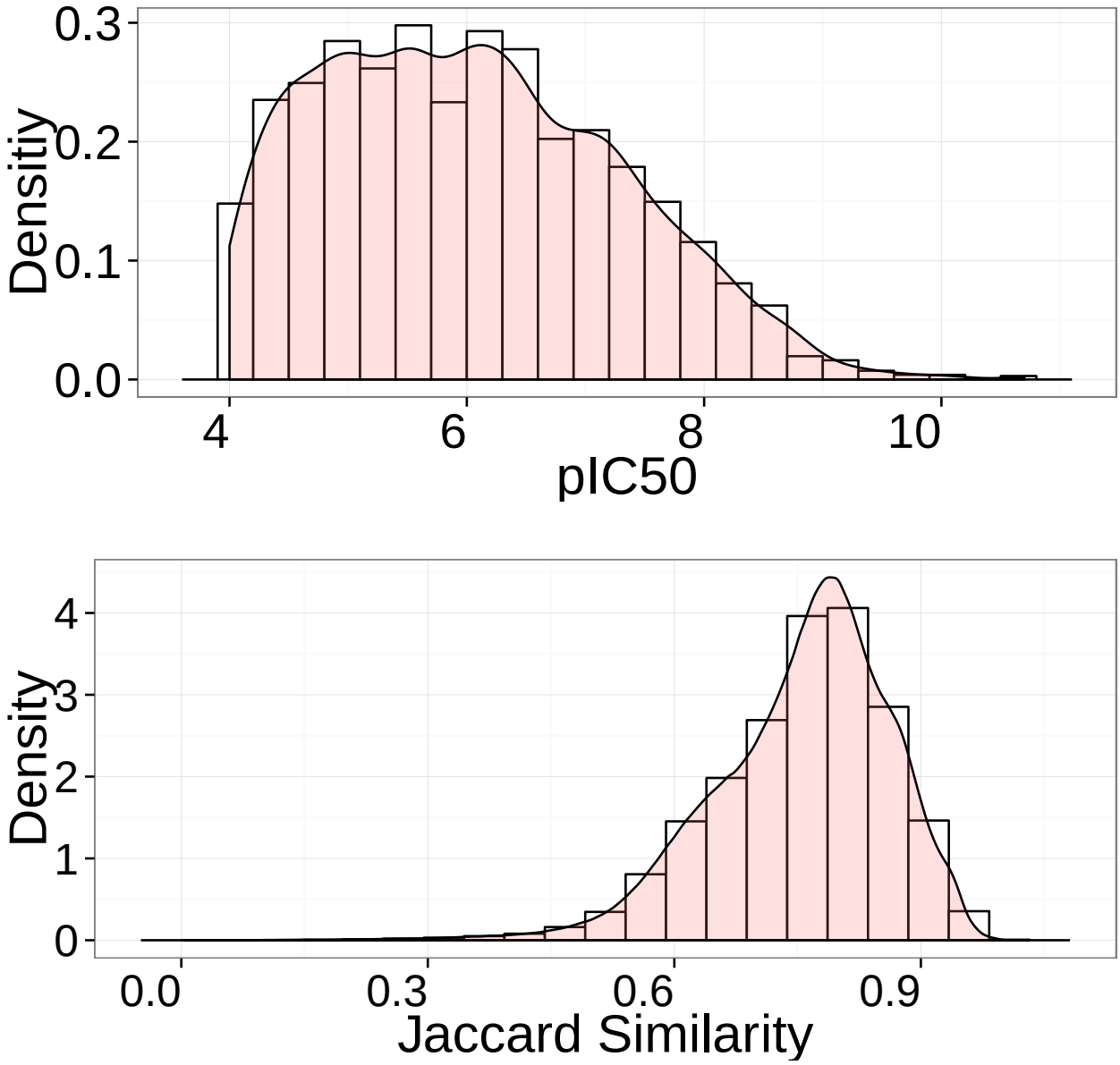

Figure 2: Density of the response variable (upper panel). Pairwise Compound Jaccard Similarity (bottom pannel)

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 a training (70%) and a hold-out (external; 30%) set 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 setin 5 folds:

```
dataset <- GetCVTrainControl(dataset, seed = 1)
saveRDS(dataset, file = "dataset_COX_preprocessed.rda")
```

All models are trained with the same CV options to allow ensemble modeling (see below). These options are: method='cv', number=folds, repeats=repeats, returnResamp='none', returnData=FALSE, savePredictions=TRUE, verboseIter=TRUE, allowParallel=TRUE and index=createMultiFolds(y.train, k=folds, times=repeats)).

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[4]. 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[5].

```
method <- "rf"
tune.grid <- expand.grid(.mtry = seq(5, 100, 5))

modelCoxRF <- train(dataset$x.train, dataset$y.train,
    method, tuneGrid = tune.grid, 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[6].

```
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, the averaged values and the fitted 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.[7]. To be considered as predictive, a model must satisfy the following criteria:[8, 9]

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 [10] and [11].
To get (i) the training, (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 ensembled:

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

Subsequently, we load the models previously trained. The list of models is in the file modelsEnsemble.

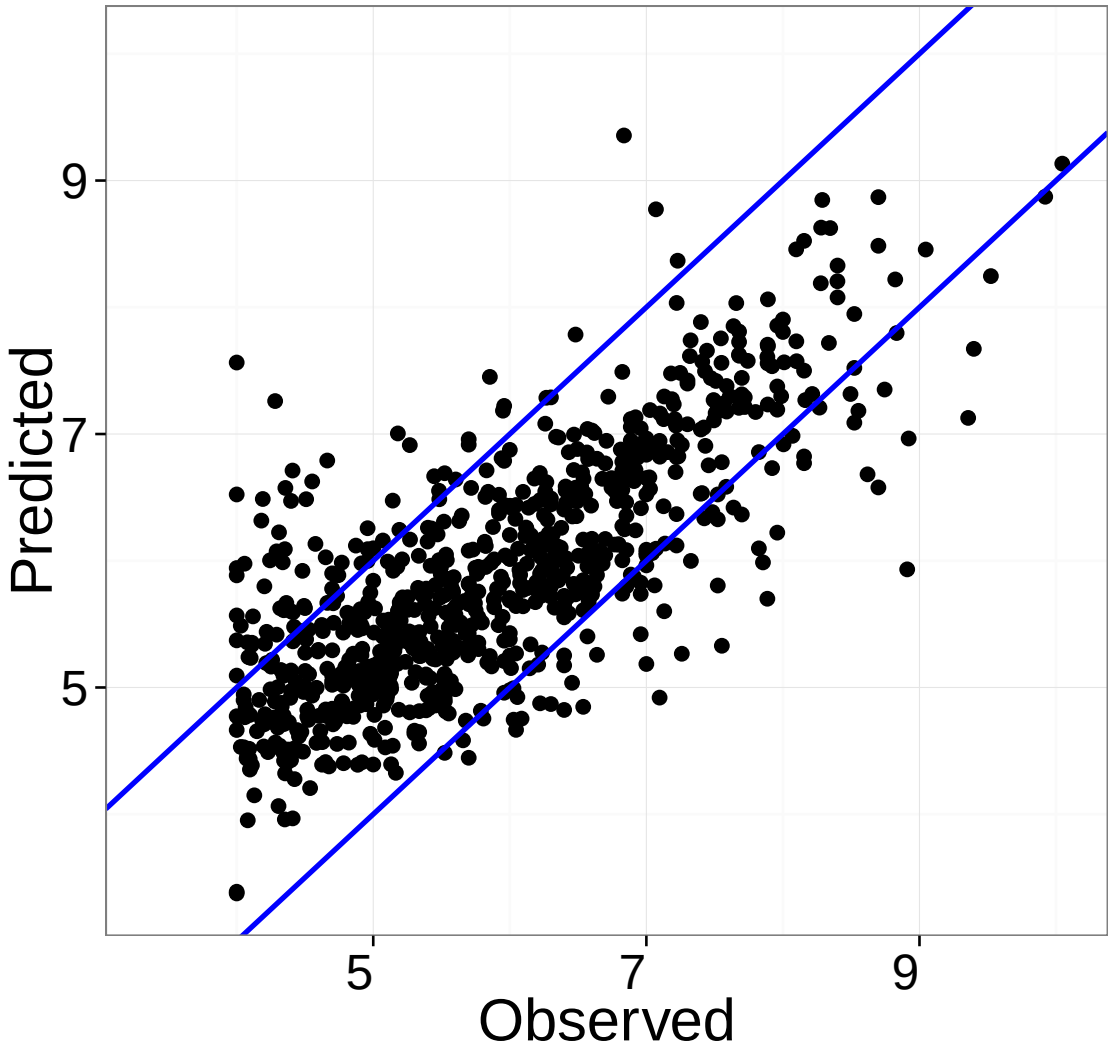

Figure 3: Observed vs Predicted

```
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 ensembled:

```
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. Malli avin, G. J. van Westen, and A. Bender, "Polypharmacology modelling using proteochemometrics: recent developments and future prospects," *Med. Chem. Comm*, 2014.

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

[4] 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, ISSN: 1553-7358.

[5] L. Breiman, "Random forests," en, *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001, ISSN: 0885-6125, 1573-0565.

[6] 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, ISSN: 0090-5364, 2168-8966.

[7] V. Consonni, D. Ballabio, and R. Todeschini, "Evaluation of model predictive ability by external validation techniques," en, *J. Chemometrics*, vol. 24, no. 3-4, pp. 194–201, 2010, ISSN: 1099-128X.

[8] A. Golbraikh and A. Tropsha, "Beware of q2!," eng, *J. Mol. Graph. Model.*, vol. 20, no. 4, pp. 269–276, Jan. 2002, PMID: 11858635, ISSN: 1093-3263.

[9] 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," en, *QSAR Comb. Sci.*, vol. 22, no. 1, pp. 69–77, 2003, ISSN: 1611-0218.

[10] I. Cortes-Ciriano, D. S. Murrell, G. J. van Westen, A. Bender, and T. Malliavin, "Ensemble modeling of cyclooxygenase inhibitors," *Manuscript in Preparation*, 2014.

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